No. 62 JULY 1983

U.S. Edition: $2.50
International Edition: $3.00

10 Major Computers Compared...See full report page 26

MICRO

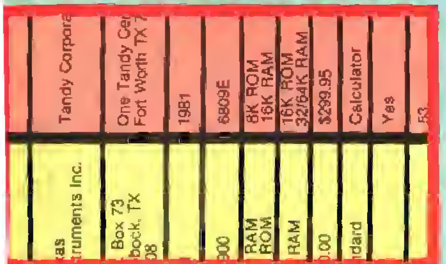
Advancing Computer Knowledge

Buyer's Guide

A Low-Cost Computer Comparison

- Apple II +, IIe
- Atari 400, 800, 1200XL
- VIC-20, C64, PET 4032
- TI 99/4A
- TRS-80C

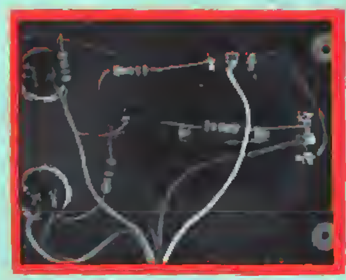
Easy-to-read chart compares graphics, keyboard, display, memory, sound, and more...



System Comparison

Pascal Solitaire

Word Detective



Apple Joystick

In this month's Learning Center:

Atari Nine-Color Painting Program

Low-Res Animation for the Apple

Simple VIC-20 and C64 Word Game

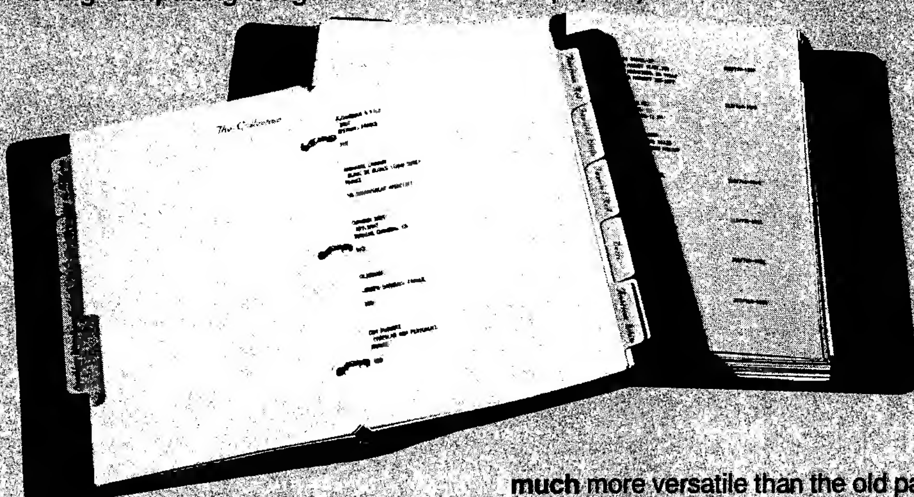
0 74470 11690 07

MAGIC MEMORY

Will Remember Everything For You

(If you want it to)

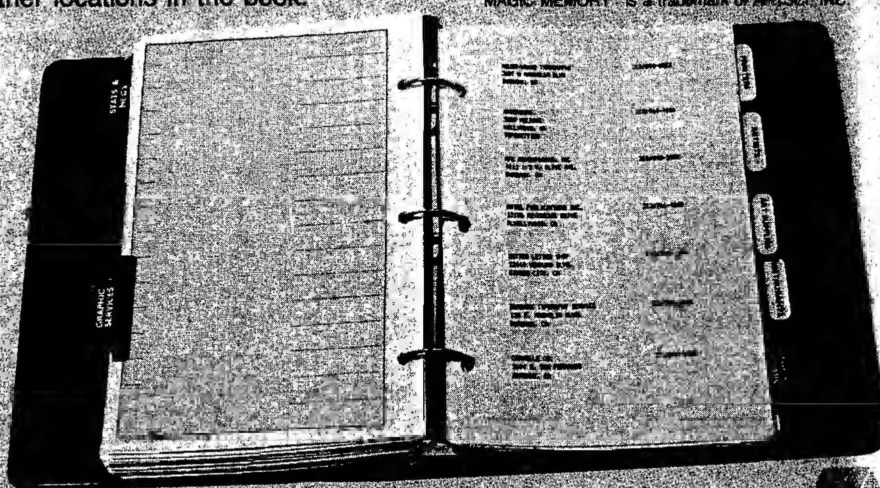
Imagine a system that would record all the wonderous, valuable information you have assimilated onto a single tiny disk. (No more scattered bits of paper, business cards, etc.) Imagine the same system giving you a typed sheet you could put into a notebook or print out for a party and instantly change, or add to, at a moments notice. Imagine cross-referencing to suit both your business needs and personal desires so that all your data was organized into one little black book! On top of all this — imagine having **fun** putting it together.



"Imagine" no more. ARTSCI, INC. proudly announces **MAGIC MEMORY™**, the complete advanced database system. **MAGIC MEMORY** is made to simplify the information storage process.

The typical first-time computer user has no *human* experience that will help him to relate to a *computer's* methods of handling information. Therefore, his learning ability is hampered and remains so for many hours of use, UNLESS the computer has been designed to run simply.

MAGIC MEMORY™ is built for the computer rookie. Everyone can relate to **MAGIC MEMORY™** because its form is familiar. It looks like an address book but its not. Its *more*. Like the address book, **MAGIC MEMORY™** presents an A thru Z index tabulation on the right edge of the video display. The user simply selects a tab and the *book* is opened to the proper page(s). A second set of tabs are available that can be labeled by the user (i.e.: companies one deals with, birthdays, lists, wines, shops, etc.) Yet **MAGIC MEMORY™** is **much** more versatile than the old paper address book. This system can instantly add or delete information, sort alphabetically, and transfer data to other locations in the book.



One of the features in the print section of **MAGIC MEMORY™** is the ability to print out any or all of the *address* book,

or any of its special sections.

MAGIC MEMORY'S data presentation screen will instantly adapt to any type of information you may want to store. It is only limited to your imagination and your needs.

MAGIC MEMORY™ is designed to operate on an APPLE //e and still remain totally compatible with APPLE II. The system will operate in 40 columns or 80 columns. You may also use the 70-column display that requires no additional hardware.

MAGIC MEMORY™ retails for $99.95. Call 213/985-5763. Use your imagination to outwit your organizational woes.

MAGIC MEMORY™ is a trademark of ARTSCI, INC.

213/985-5763
5547 Satsuma Avenue
North Hollywood, Ca 91601

artsci

Circle No. 1

MICRO™
Highlights

About the Cover

The graphic in our screen was created with Paul Swanson's Atari painting program (page 66).

Do not panic at the words "Hardware Feature." You need not be totally hardware-oriented to enjoy the July issue of MICRO. Our lead article entitled "Low-Cost System Comparison" (pg. 26), by Phil Daley and Loren Wright, is a descriptive comparison of some of the more popular low-cost systems available. Then Charles Putney describes a simple modification and interrupt-driven program to allow use of 1/60-second interrupts on the Apple. See "A Clock Interrupt for Your Apple" (pg. 36).

The joystick is a popular hardware item to modify. We have two articles that will improve your joystick capabilities. For the Color Computer, John Steiner shows you how to use Atari joysticks on your machine (pg. 42). For the Apple, Dan Weston explains how you can construct your own joystick for the Apple II at a savings of over 50%. Read "An Inexpensive Joystick for the Apple II" (pg. 48).

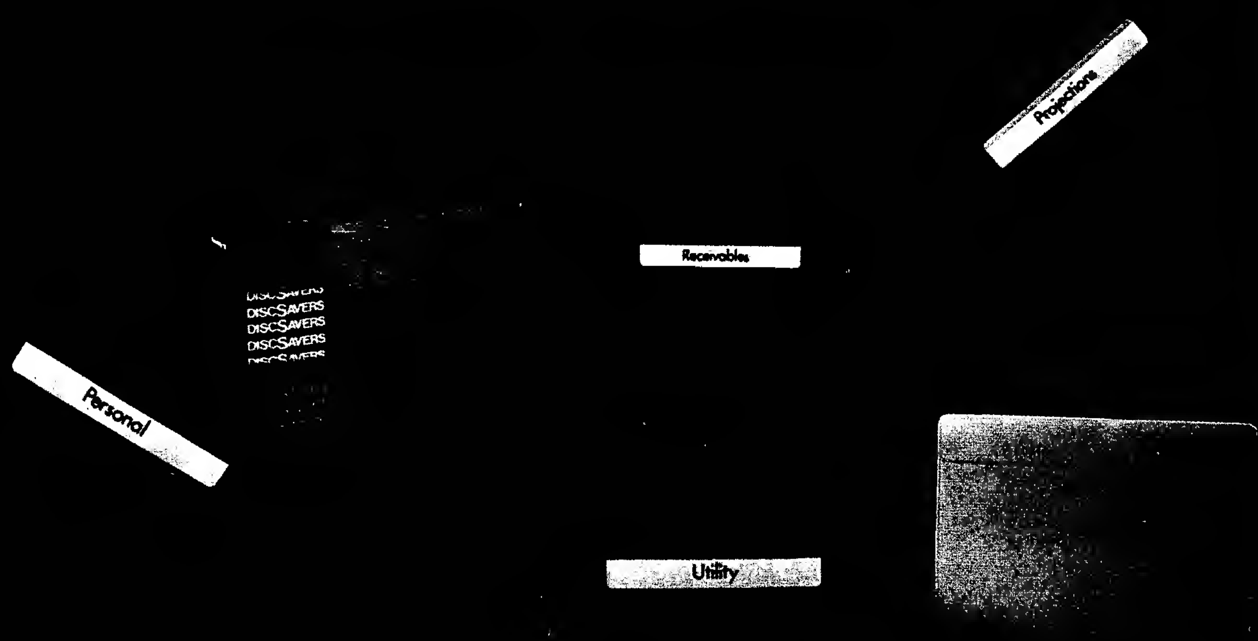
Jerry D. Brinson gives you directions for constructing an EPROM Programmer. Included in his article "PET Goes ROM" (pg. 50) is software necessary to drive the Programmer on a Commodore PET 2001 computer. And in "Automatic Head-Load Control for OS-65D" (pg. 54) Peter Kleijnjan explains how a few bytes of patch code added to the operating system can give the minifloppy system user a head that is loaded only during disk activity.

Our final article in the feature section is for the experienced hardware enthusiast. "Disk Interface for Single Board 6502's" (pg. 56) by Jack Brindle is complete with hardware diagrams and sample software drives that interface a floppy disk to a 6502-based system.

MICRO™

DISCSAVERS

VINYL PROTECTIVE DISK SLEEVES



COLOR CODED: Multi-color DiscSavers™ are designed for easy recognition of individual disks with your own color-keyed filing system. Ideal for office or home use.

PROTECTIVE: Custom grain vinyl provides added protection for magnetic disks by guarding against common handling hazards.

ATTRACTIVE: DiscSavers provide a handsome and professional method of single disk storage and enhance the look of your hardware while protecting your valuable software.

DURABLE: Rigid vinyl construction protects against constant handling to ensure long wear and tear.

PORTABLE: DiscSavers are the only portable vinyl disk sleeves for use with a single diskette that bear the RockRoy mark of quality.

Contact your Dealer or Distributor

ROCKROY INC.
Computer Products Division

7721 E. Gray Road
Scottsdale, Arizona 85260
(602) 998-1577
Toll-Free 800-528-2361

Circle No. 1

DiscSavers is a trademark of RockRoy Inc.

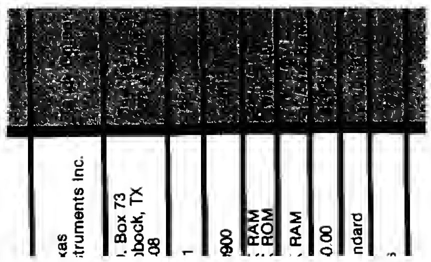
MICRO™

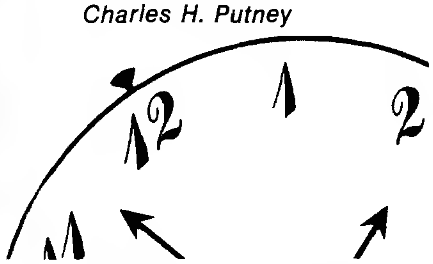
Advancing Computer Knowledge

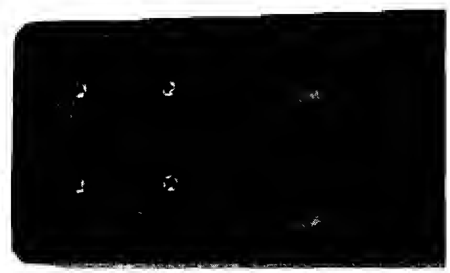
MICRO
P.O. Box 6502,
Amherst, NH 03031
(603) 889-4330

MICRO is published monthly by: MICRO, Amherst, NH 03031. Second Class postage paid at: Amherst, NH 03031 and additional mailing offices. USPS Publication Number: 483470. ISSN: 0271-9002. Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to MICRO, 10 Northern Blvd., P.O. Box 6502, Amherst, NH 03031, or call (603) 889-4330, Telex: 955329 TLX SRVC, 800-227-1617. *Subscription rates (per year):* U.S. $24.00, $42.00 / 2 yr. Foreign surface mail $27.00. Air mail: Europe $42.00; Mexico, Central America, Middle East, North Africa, Central Africa $48.00; South America, South Africa, Far East, Australasia, New Zealand $72.00.

Hardware Feature







Atari Painting Program...page 66

The Learning Center

Articles

Columns

Departments

MICRO™

Editorial

The Peanut Butter Test

I recently had the opportunity to attend a symposium entitled "The Small Systems Economy: Crisis and Opportunity." Several speakers offered valid and interesting comments on the microcomputer industry, but one particular topic — covered by former president of Commodore, H. E. James Finke — caught my attention.

Mr. Finke pointed out that the curve of technology *versus* price had converged on a figure of about $200.00, a sum the consumer seems willing to pay for a microcomputer. Of course, we know that considerable discounts are offered on many systems; e.g., you can buy a VIC-20 for $88.00 in some department stores. In any case, the new low prices mean that more people are able to buy computers — including those who know nothing about them.

As you can see, at least two problems could arise from the low price and new forms of distribution. First let's address the aspect of new users. Most of these customer's have probably never seen, much less used, a computer before. They won't know that their machine needs a little more care and consideration than, say, their TV or toaster oven. And chances are they won't be as careful with equipment that cost $88.00 as with a $1000.00 system. Hence, we come to what Finke terms the "peanut butter test."

Should manufacturers produce computers that will hold up to the abuse of sticky fingers, spilled milk, and cookie crumbs? Since the lower prices are attracting more people and a new market (a market most microcomputer manufacturers are trying very hard to capture), perhaps the computers should be especially durable. How about crumb trays underneath the keyboard? Or a waterproof casing? Maybe an attachment to hold drink glasses would be convenient (you know, the kind you put your coffee in while you commute to work).

Of course these suggestions may sound outrageous, but there are, no doubt, more reasonable alternatives. The point is, now many microcomputers are not only priced for mass consumption, they are more easily available. Instead of being marketed in computer stores or electronics centers, they are being sold in department stores by clerks who know very little about them — to customers who know even less. These systems are bound to encounter a lot more rough treatment than ever before. This brings us to the second problem: proper training for sales people selling micros.

I would like to suggest that it is time for manufacturers to provide serious training support for their distributors. Some manufacturers are working closely with dealers in computer stores, providing instruction for the store sales staff. But what about the clerks in the general department stores? Not only do most of these sales clerks lack sufficient knowledge to effectively sell a VIC-20, TI-99/4A, or Atari, but quite often there isn't enough instructional material at the store for an adequate demonstration. If customers are going to purchase a microcomputer that they know little about operating, they certainly won't know its physical limitations.

It seems to be up to the manufacturers to provide both the training for the distributors and literature for the public. Meanwhile, in lieu of actual hardware modifications that offer more durability for a system, perhaps flyers could be passed out with each computer explaining care and maintenance dos and don'ts. For instance, "Do not leave drinks on or near the machine. Disks should not be put on the stove. The keyboard will not operate properly when caked with peanut butter......."

Marjorie T. Morse

Marjorie Morse
Managing Editor

MICRO™

Letterbox

Opinions on APL

Dear Editor:

I was pleased to notice Terry Peterson's article on APL in MICRO #57.

Mr. Petersen might have found that, in the comparison of APL with BASIC he ran on his SuperPET, APL would fare even better had he used the simplest possible APL command to generate an array containing the first 1000 integers (namely A←ι1000). On the other hand, APL would probably have demonstrated less of a timing advantage in generating an array consisting of the first 1000 EVEN integers (A←2×ι1000), although the advantages of compactness and clarity are still present.

The issues of structure and readability raised in the article are certainly very live concerns in the world of APL programming, but are essentially distinct. Briefly, program structure is largely irrelevant in small and medium-sized APL applications because a straightforward non-branching approach is almost always best. Structuring techniques can certainly be applied when needed, however (see Geller, D.P. and Freedman, D.P., *Structured Programming in APL*, Winthrop Publishers, Cambridge MA, 1976). Readability, on the other hand, is enhanceable by such techniques as careful choice of names, use of modular code, and adherence to a set of simple programming standards. Good APL code can be very readable indeed!

It is probably the case that, as Mr. Peterson discovered, the documentation supplied with a microcomputer implementation of APL will not generally suffice to teach the idea of the language. Even though new users of the language will find that they can immediately perform simple but significant computations in APL, they will need to consult further articles and texts and spend a lot of time at the keyboard in order to command the real power of the language (a worthy goal, however!). As yet there do not seem to be any texts available on the general marketplace that teach APL in the context of the microcomputer, but there are numerous good mainframe-oriented books available to the novice.

R.W.W. Taylor
967 Meigs St.
Rochester, NY 14620

Here is Mr. Peterson's response:

1. Regarding the need to print APL characters I offer a solution in an article, accepted for publication in MICRO, that tells how to print APL text on an Epson MX-80 with Graphtrax.

2. On my interpreter speed comparison Mr. Taylor is obviously right: You don't need to 'reshape' a 1000-element vector into a 1000-element vector. (The blundering of a beginner!) Using his suggestion cuts the APL benchmark run time by about 10%. He is also correct about the even-integer assignment, which takes about three times (!) as long as the simple integer assignment on the SuperPET.

3. On the issue of structure and readability I strongly disagree with Mr. Taylor's remark that structure is *essentially* distinct from readability. As far as I can see the principal advantages usually claimed structured programming boil down to generating more readable code. Such code is easier to debug and maintain (i.e., modify). As I tried to indicate in my article, the 'correct' APL approach to most problems involves fewer loops and branches than would arise in other languages. However, the proper APL solution to even medium-sized problems may require more RAM than is available in the SuperPET. In these cases, the non-branching APL solution to a programming problem is unworkable and the lack of more modern branch statements is sorely felt. I remain unpersuaded that APL can be 'structured' by merely pointing out topological equivalences to other languages. Just because I might be able to construct in old-fashioned BASIC an 'IF × THEN 1234' code sequence that is 'just like' (i.e., equivalent in program flow to) Pascal's if × then ... else ...' structure doesn't make such code as readable as Pascal! Futhermore, I assert it is not meaningfully 'structured'. The proper

(Continued on page 10)

SAVE 20%!!!!

Subscribe to MICRO™

Use This Postage Paid Card to Order the Next 12 Issues of MICRO and SAVE $6.00 Off Newsstand Price!

MICRO™ SAVE 20%

☐ Yes! Enter my subscription to MICRO, and send me the next 12 issues for just **$24.00.** I save **$6.00 off** the newsstand price!

Name ____________________

Address ____________________

City ____________________

State ____________________ Zip ____________

I'm paying by ☐ Check ☐ MO ☐ VISA ☐ MasterCard

Card # ____________________ Exp.Date ____________

Signature ____________________

I OWN A:
☐ Commodore 64 ☐ VIC-20
☐ Apple ☐ PET
☐ Atari 400 ☐ Atari 800
☐ Other: ____________

For Faster Service Call:
1-800-345-8112
(In PA: 1-800-662-2444
VISA or MasterCard Only

A Feast Of Computing Ideas...

Order These Books From MICRO™

MICRO™ Books

NEW for VIC-20 Users!
Mastering Your VIC-20
With 8 BASIC Projects

A book that makes learning to program your VIC-20 fun! Contains 8 projects and programs. Games, utilities — even a VIC-20 version of "VisiCalc." All 8 programs on cassette to help you learn faster.

☐ Mastering the VIC-20 @ $23.95

NEW for OSI Users!
MICRO on the OSI

Includes Machine-Language enhancements and BASIC Aids, hardware modifications for enhanced/reversed video, programs for control code and upper/lower case entry. A valuable programming tool.

☐ Micro on the OSI @ $19.95

Best Sellers for APPLE Users!
MICRO on the APPLE

Programming aids, utilities, games, enhancements. Together Volumes 1, 2, and 3 contain over 100 programs on diskette. Fully documented and explained.

☐ 3-Volume Gift-Boxed @ $59.95
☐ Vol.1☐ Vol.2☐ Vol.3 $24.95 ea.

Please rush the MICRO Books I've checked above to:

Name ____________________

Address ____________________

City ____________________ State ________ Zip ________

(Allow 6-8 weeks for delivery)

I'm paying by: ☐ Check ☐ MO ☐ VISA ☐ MC

Total Enclosed: $ ____________
(Add $2.00 s/h per book. MA res. add 5% tax)

Visa/MC # ____________________

Exp. Date: ____________________

MICRO™ Reader Service...

For More Information On The Advertisers In This Issue!

MICRO™ Reader Service Card

The numbers below correspond to the numbers imprinted on the advertisements in this issue of MICRO. This card valid for 90 days only.

Mail information to:

Name ____________________

Address ____________________

City ____________________ State ____________ Zip ____________

1	2	3	4	5	6	7	8	9	10	11	12	13
14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52
53	54	55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76	77	78
79	80	81	82	83	84	85	86	87	88	89	90	91
92	93	94	95	96	97	98	99	100	101	102	103	104
											105	106

July 1983

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 20, AMHERST, NH

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

10 Northern Blvd.
P.O. Box 6502
Amherst, NH 03031

SAVE 20%!!!!

Subscribe to MICRO

Use This Postage Paid Card to Order the Next 12 Issues of MICRO and SAVE $6.00 Off Newsstand Price!

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 20, AMHERST, NH

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

10 Northern Blvd.
P.O. Box 6502
Amherst, NH 03031

A Feast Of Computing Ideas...

Order These Books From MICRO

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 20, AMHERST, NH

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

10 Northern Blvd.
P.O. Box 6502
Amherst, NH 03031

MICRO Reader Service...

For More Information On The Advertisers In This Issue!

☐ APPLE ☐ EAGLE ☐ IBM, PC ☐ NORTHSTAR ☐ TRS-80 II 8" ☐ OSBORNE ☐ FRANKLIN ☐ STANDARD CP/M 8" ☐

☐ XEROX 820 ☐ GAMES ☐ COMPILERS ☐ UTILITIES ☐ BUSINESS ☐ ACCOUNTING ☐ SPREAD SHEETS ☐ MEDICAL ☐ DENTAL ☐ PAYROLL ☐ GENERAL LEDGER

RENT SOFTWARE BEFORE YOU BUY!

from our

SOFTWARE RENTAL LIBRARY

You can now RENT the most popular software available for just 20-25%* of Manufacturers' Retail Price

- Eliminate the risk—rent first!
- 100% of rental fee applies toward purchase
- All purchases are 20% Off of Manufacturer's Suggested List
- Rentals are for 7-days (plus 3 days grace for return shipping)

SPECIAL INTRODUCTORY OFFER

There are now 2 different plans to choose from:

Join the **Game Group, $25.00** per year and rent as many games as you like for only 20-25% of Mfrs. Sugg. Retail Price.* Minimum order, 3 game rentals	Join the **Business Group, $50.00** per year and rent as many business application programs as you like for only 20-25% of Mfrs. Sugg. Retail Price.*

REMEMBER, THESE ARE NOT DEMOS, BUT ORIGINAL UNRESTRICTED SOFTWARE PROGRAMS

(complete with manuals in original manufacturers' packages)

To Immediately Order, or for more information:

UNITED COMPUTER CORP.
Software Rental Library
Culver City, California

Toll Free CALL 1-800 992-7777
In California CALL 1-800 992-8888
In L.A. County CALL 1-213 823-4400

Canadian Orders Welcome

AMERICAN EXPRESS

*Plus postage and handling

☐ EDUCATIONAL ☐ ACCOUNTS RECEIVABLE ☐ WORD PROCESSORS ☐ DATA BASES ☐ LANGUAGES ☐ GRAPHICS ☐

☐ ACCOUNTS PAYABLE ☐ MAILING ☐ SPELLING ☐ COMMUNICATIONS ☐ REAL ESTATE ☐ TAXES ☐ STOCK MARKET ☐ INVENTORIES ☐ HEATH/ZENITH 89 8"

approach is to do as Waterloo Computer Systems did with microBASIC on the SuperPET — add 'if...then...else...', etc., to the language.

Aesthetically, I think APL is a close relative of haiku poetry. Both are able to express very much very compactly. And, with study, both are beautiful. Unfortunately, APL's elegance comes with a sacrifice in efficiency, since not all data structures are naturally represented as rectangular arrays.

Terry M. Peterson
8628 Edgehill Ct.
El Cerrito, CA 94530

Feedback on Newton-Raphson

P. P. Ong's program, "Extending Newton-Raphson's Method to Evaluate Complex Roots" (56:71) can be speeded up by using Horner's algorithm (the factorization of the polynomial). Following are the lines that replace the original ones:

```
51200  R = A(N) × COS (N × TH)
51220  FOR I = N - 1 TO 0 STEP - 1:
       R = A(I) × COS (I × TH) + R
       × A : NEXT
51240  S = A(N) × SIN (N × TH)
51260  FOR I = N - 1 TO 1 STEP - 1 :
       S = A(I) × SIN (I × TH) +
       A × S : NEXT
51280  T = N × A (N) × COS ((N - 1)
       × TH)
51300  FOR I = N - 1 TO 1 STEP - 1 :
       T = I × A (I) × COS ((I - 1) ×
       TH) + A × T : NEXT
51320  U = N × A (N) × SIN ((N - 1)
       × TH)
51340  FOR I = N - 1 TO 2 STEP - 1
       : U = I × A(I) × SIN ((I - 1)
       × TH) + A × U : NEXT
```

Lic. Victor L. P. Frank
Argentina

Dear Editor:

I enjoyed reading Dr. Ong's article "Extending Newton-Raphson's Method to Evaluate Complex Roots" (56:71), but I believe his program is not that accurate.

Although finding the roots of a high order polynomial can be difficult, checking the result is easy — just substitute back into the original equation.

In the example (eqn. 22), Dr. Ong gives the real roots as 0.5100436 and 2.1458455. These roots are actually 0.509721 and 2.307073. The error produced by the program in calculating the root at 2.307073 is over 7%, which is much larger than the "approximately 0.0001%" stated by Dr. Ong.

Peter Chrzanowski
115 N. Sunset Dr.
Ithaca, NY 14850

Dr. Ong responds:

Of course Newton-Raphson's method is not new, but its extension to include complex roots is novel. The algorithm I've proposed is just suitable for the present-day microcomputer (or even a programmable calculator) environment. I also agree with Lic. Victor that his method of evaluating the four polynomials is shorter. However, in choosing the procedure presented I have intentionally sacrificed speed for clarity. In fact, there are many more ways to shorten computation time, computational steps or even program length, but as the program was primarily intended for the "average" (and probably busy) reader to follow without too much effort, I have made no attempt to refine the program at the expense of readability.

Peter Chrzanowski's point is well-noted. The 7% discrepancy was actually due to a missing term of $-6X^{10}$ in equation (21) or $+6X^{10}$ in equation (22). Unfortunately the omission was discovered only after the manuscript went to press. The accuracy of 0.0001% still holds. If Peter had run the program using the uncorrected equation (22) as input, he would no doubt find that it yields the following roots:

$x = 0.50972140$
$0.33224526 \pm 0.44192513i$
$0.47405998 \pm 1.00831167i$
$-0.38559764 \pm 1.04245339i$

2.30707296
$0.98920983 \pm 0.39823923i$
$-0.99690848 \pm 0.36963689i$
$-1.65691560 \pm 1.7025900li$

The relevant answers are more than one significant digits better than those quoted by him.

I am glad that the program did not make any error, only the human part of me did.

Dr. P. P. Ong
Physics Department
National University of Singapore
Singapore

Updates and Microbes

A New Look at a Full Byte

Mark J. Boyd offered his solution to the "lack of Apple graphics" problem when using the Epson MX printer ("A Full Byte for Your Apple Printer" 58:42). Dr. Boyd used an annunciator output to control bit 7 of the Apple output to the Epson printer. However, I believe it is not the best solution to the problem. My solution is a sort of double reverse.

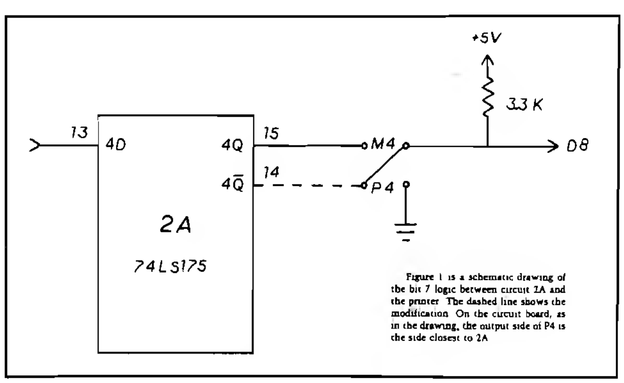


Figure 1 is a schematic drawing of the bit 7 logic between circuit 2A and the printer The dashed line shows the modification On the circuit board, as in the drawing, the output side of P4 is the side closest to 2A

Figure 1

(Continued on next page)

Updates and Microbes

The Epson interface uses two 74LS175 4-bit latches to buffer data on it's way to the printer. If jump P4 is in place the value latched at 4Q is ignored and the output bit is held low. If instead jumper M4 is in place the value latched at 4Q is passed.

It occurred to me that what we really want here is for the INVERSE of 4Q to be sent to the printer. In that case, Apple characters with their high bits set will be properly sent to the printer with their high bits clear. On the other hand, all we have to do to access the graphics is send out characters with their high bits clear and the interface will translate them to printer graphics characters.

Make sure that both jumpers P4 and M4 are removed. With the component side of the board facing you, carefully solder a piece of small gauge insulated wire (wire-wrap wire is ideal) between pin 14 of circuit 2A and the output line at P4 or M4. For P4 the output is the hole closest to circuit 2A and for M4 it is the hole farthest away. (See figure 1)

Owing to the contrariness of the Apple's COUT monitor routine, we need a special driver to send out characters with their high bits cleared. The following COUT routine will allow unaltered transmission of characters to the printer. The routine is used by POKEing the ASCII character value into $300 and then CALLing $301. It is assumed that the interface is in slot 1. Otherwise the addresses will have to be altered accordingly.

```
301   AD 00 03   COUT   LDA $0300
304   2C C1 C1   BUSY   BIT $C1C1
307   30 FB             BMI BUSY
309   8D 90 C0          STA $C090
30C   60                RTS
```

New MICRO Bulletin Board Number!

You can reach our Bulletin Board now at (603) 883-1576.

MASTER Improvement

In MICRO 59:80 change line 2050 as shown to improve the flashing of the cursor: 2050 POKE CF,0: POKE CF + 3,0.

MICRO

MICRO

PET Vet

Loren Wright

FORTH for the Commodore 64

Alternate languages for the PET, no matter how powerful, how fast, or how memory-efficient, have never been able to mount a serious challenge to BASIC. I have reported in this column on several excellent languages: FORTH (45:55), Pascal (44:50), RPL (43:35), and COMAL (46:98). All of these are more powerful and some are faster than BASIC, but none has ever completely overcome the fact that, even when the new language is in control, the BASIC ROMs just sit there taking up memory space. RPL probably does the best in this respect since it makes heavy use of the ROM routines by keeping down redundant memory use. COMAL and Pascal rely rather heavily on the disk to keep things manageable. The problem, of course, is in the design of the PET. The ROMs are soldered directly into the board, thereby permanently staking out 14–18K of valuable memory. Only with hardware modifications or add-on boards can this memory be reclaimed.

The design of the C64 allows for a number of things, including swapping out the BASIC ROMs to make RAM available in their place. The I/O and Kernal ROMs are independently selectable, so an alternate language doesn't have to write new routines to do the more mundane things. That is precisely what has been done with *C64 FORTH* (Performance Micro Products, 770 Dedham Street-S2, Canton, MA 02021). All the memory from $800 to $CFFF is available for FORTH, its dictionaries, and its buffers. Many of the new sophisticated cartridge games are based on *C64 FORTH*. There are reasons, besides memory conservation, that make FORTH an ideal tool for development of sophisticated software for the C64.

One major advantage is speed. I rewrote some of the BASIC sound demonstrations from the *Programmer's Reference Guide* with *C64 FORTH*. At first I thought that the language was a failure because the effects didn't sound anything like their BASIC equivalents. The error, though, was in my translation of the delay loops: where BASIC used FOR I=1 TO 1000, FORTH requires 6001 1 DO LOOP for the same delay. Other operations, such as setting or clearing pixels in a high-resolution screen, are also much faster. (You should see the sprites whipping across the screen!) The only thing faster is straight machine language, but writing assembly language, even with a powerful assembler like MAE or the Commodore assembler, is a slow process. By the way, there is an assembler available on the system disk, so if FORTH isn't fast enough for something, you can code it directly in assembly language anyway!

Other advantages are convenience and program understanding. Those of you with some experience programming sound and graphics on the C64 know the great number of POKE instructions required to do something as simple as sounding a middle C. For instance, in FORTH you can define a word that performs all the initialization

for music. You could define others that set up the envelope, filters, and other parameters for different musical instruments. In fact, you could define a whole series of special music-oriented words and then resave FORTH with these words included in the vocabulary. When you reload your special music-FORTH, the music words are available immediately. As you look at the listing, the words help you to understand what you programmed. In a BASIC program listing, a series of POKE and GOSUB instructions would have to be reinterpreted.

I should point out that FORTH listings can be much less understandable than BASIC listings and that even the simplest FORTH programs are meaningless to inexperienced FORTH programmers. The reason why FORTH listings make no sense at first is in the way they are designed; this design is the source of most of the power of the language. For an experienced FORTH programmer, the listings have the potential to be much more meaningful.

FORTH uses Reverse Polish Notation (RPN) and the language is heavily oriented toward a user *stack*. Hewlett-Packard calculators use both RPN and a user stack. Pascal and BASIC also use stacks, but the user usually doesn't even know about it. A stack is essentially a pile with items *push*ed onto it or *pull*ed from it. Each item is a 16-bit number and most FORTH words operate from the top of the stack. To perform 5 + 3, you push 5 on the stack, followed by 3. The word '+' adds the top two numbers and replaces them with their sum. Pressing RETURN after entering the following will accomplish this:

5 3 + . *8 OK*

The period prints the number on the top of the stack, and the italics indicate what the computer prints in response. The expression (5 + 3) × (9 – 2) is done with the following sequence:

5 3 + 9 2 - * . *56 OK*

Not all operations can be performed from the top of the stack, so there are words to access items beneath. There are also words that consider items in 32-bit increments (*double* numbers). Others do such things as comparisons, looping, and program control. There are many more, and I can't even begin to scratch the surface.

One of the greatest powers of FORTH is its ability to expand its vocabulary. The language accomplishes this with a structure called a *colon definition*. It begins with a colon, followed by the name, the sequence of already defined FORTH words and numbers, and a semicolon. Here are some handy words for C64 graphics:

```
: CLRCOLOR 55296 1000 ROT FILL ;
```
(Sets all of color memory to color number on top of stack)

(Continued on next page)

PET VET *(continued)*

```
:  CLEAR 147 EMIT ;
   (Clear screen)
:  SCREEN 53281 C! ;
   (Set screen color with number on top of stack)
```

FORTH programming is done in a rather unusual fashion. Instead of using 'top down' programming, it uses a 'bottom up' approach. You start by defining the simplest words, building on these successively, until you can write your program in relatively few, already defined words.

Now that I've convinced you that FORTH is a good thing, I should tell you about a few of its deficiencies. The biggest deficiency is the lack of floating-point capability. Also, exponentiation, logarithms, and trig functions are lacking. These functions can be added to the vocabulary, if needed. (An advantage of FORTH is that it is one of the most standardized languages available.) If you need floating-point routines, you can probably find them published somewhere and easily adapt them to your particular machine. Work is already underway at Performance Micro Products on a package that will include floating point, plus all the other BASIC functions.

I've spent a lot of space talking about FORTH in general, but very little on *C64 FORTH* from Performance Micro Products. That's because you have to know something about FORTH to understand an evaluation and also because FORTHs are mostly identical due to standardization. There are two standards — fig-FORTH and FORTH-79 STANDARD. *C64 FORTH* is of the latter type, a more elaborate version. In addition, author Gregg Harris has provided a number of convenient words that apply specifically to the C64 system. Two of FORTH's weaknesses have been overcome. FORTH screens are saved as regular CBM data files, which means you can mix BASIC and FORTH files on a disk. It also means that CBM DOS can be used to full advantage for copying, renaming, etc. The other problem is with the editor. The standard FORTH editor is a line editor. Some people love line editors; I don't. Gregg has written a very nice screen editor that allows full use of cursor controls and adds such commands as insert-line, delete-with-save, and un-delete. I would add tabs, but otherwise I found the editor very convenient.

Software developers will find the SAVETURNKEY word a necessity. It saves the current application so that it will run automatically on loading. The user does not end up with a usable system, so the developer is not violating the copyright. Another handy feature is a *trace* function. Each word or literal, as it is encountered, is shown along with the current stack contents, which makes it easy to debug your definitions.

The manual includes a lot of useful information: a brief introduction to FORTH; description of the assembler, the

Design & Illustration: Paul Stephenson

ADVENTURE.
THE KEY IS YOUR COMPUTER

KONGO KONG

Climb ladders, avoid the barrels the crazy ape is rolling at you, and rescue the damsel. Fast machine code action.

Available for VIC-20 and COMMODORE 64

PLAYED WITH JOYSTICK OR KEYBOARD

$19.95

GRAVE ROBBERS

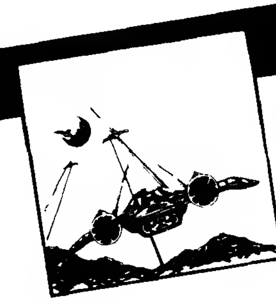
Introducing the first GRAPHIC ADVENTURE ever available on the VIC-20! With realistic audio-visual effects, you explore an old deserted graveyard and actually see the perils that lie beyond.

Available for VIC-20 and COMMODORE 64

PLAYED WITH KEYBOARD

$14.95

ANNIHILATOR



Defend your planet against the hostile aliens. All machine code makes this "Defender-like" program one of our best arcade games.

Available for VIC-20 and COMMODORE 64

PLAYED WITH JOYSTICK

$19.95

ADVENTURE PACK I

(3 programs)

Jack and the Beanstalk Defeat the Giant in your quest for his Golden Hen.

Computer Adventure Re-live the "excitement" of getting your computer. An adventure with a very different flavor.

Moon Base Alpha You must find a way to destroy the meteor that is racing towards your base, or else all moon colonies will be demolished!

Available for VIC-20 and COMMODORE 64

PLAYED WITH KEYBOARD

$19.95

ADVENTURE PACK II

(3 programs)

African Escape As the sole survivor of a plane crash, you must find your way out of the dark continent.

Hospital Adventure You are a spy whose mission is to complete the bungled assassination attempt on the evil dictator, who is recuperating in the hospital under heavy guard.

Bomb Threat Get back to town to warn the authorities of the bomb planted by the terrorists who left you prisoner at their hideout.

Available for VIC 20 and COMMODORE 64

PLAYED WITH KEYBOARD

$19.95

CHOMPER MAN

High speed machine action. Don't let the bullies catch you in a game packed full of machine code.

Available for COMMODORE 64

PLAYED WITH JOYSTICK OR KEYBOARD

$19.95

Programs for the **VIC-20** and the **COMMODORE 64**

All programs fit in standard memory. All programs are on cassette tape.

VICTORY SOFTWARE®

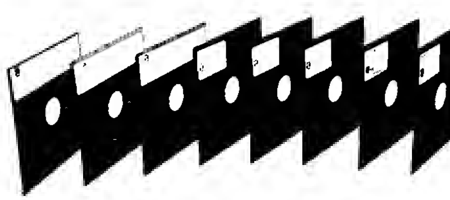
VICTORY SOFTWARE INC.
7 Valley Brook Road, Paoli, PA 19301
(215) 296-3787

Check your local dealer or order directly.
ORDERING We accept personal checks, money orders, VISA, and MASTER-CHARGE. Charge orders please include number and expiration date. Overseas orders please use charge or have check payable through a U.S. bank. Add $1.50 postage and handling per order. PA residents please add 6% sales tax.

VISA

master charge

Circle No. 15



PET VET *(continued)*

editor, and some of the special C64 system words; a section on C64 graphics, including sample routines and demonstrations (that should have been recorded on the disk); a section on I/O and use of buffers; and a full glossary. Unfortunately, there is no index or table of contents, so I found myself doing a lot of page turning. If you know nothing about either FORTH or the Commodore 64, you will get confused very quickly. The graphics examples work well, but you have to know what you are doing to enter the right words in the right sequence.

At first I had a little trouble trying to use *C64 FORTH* with my *C64 Link*. A simple POKE instruction executed after loading the program and before running it did the trick. It leaves the *Link* right in the middle of the memory space, where only large applications will cause a problem. Even this can be overcome with the appropriate relocator routine.

I recommend *C64 FORTH* for anyone about to embark on a major software development project for the C64. It will make the project go much faster. If your programming will be mostly for your own benefit, and you don't already know anything about FORTH, I suggest waiting for the FORTH cartridge by Tom Zimmer from Human Engineered Software. This version will be very much extended to include convenient words for sound and graphics. Because it is cartridge-based, it will not be suitable for development of saleable products. The HES FORTH for the C64 should be available in July or August. Performance Micro Products also has an extended cartridge version planned, available about the same time this summer.

Update on June Column

The list price for *Script 64* is now $99.95. The full name of the word processor sold by Professional Software is *WordPro 3 Plus/64*. *WordPro 3* and *WordPro 3 Plus* are different programs (for the PET) and should not be mistaken for the Commodore 64 version.

MICRO

From Here To Atari

by Paul Swanson

Many readers have sent in questions concerning use of the Atari 850 interface. Larry L. Farmer in Dover, Delaware, asked about the bootable handler for the 850.

To use the 850 just for the printer is no problem — make sure it is off when you boot the computer and turn it on when you want to use the printer. Use the disk normally, as if the 850 weren't there. There is no bootable handler required for using the printer through the 850.

If you want to use the other four ports on the 850, you do need a handler. The handler loader is in the AUTORUN.SYS file on your DOS disk. You do need the handler for modem programs, for example. Using Atari's Telelink I requires that you have the 850 on when you boot the computer and it will load the handler. Don't Ask Software's TeleTari works the same way, except it uses the disk instead of a cartridge.

To use the AMODEM programs or other communication programs, or your own software to access devices through these ports, boot your computer with the DOS disk and make sure the 850 is on when you boot it. When it loads the handler you will hear a tone from the TV speaker a second or two in length. That is the handler loading from the interface.

The handler is located and booted from the interface. It controls device R (R1, R2, R3 and R4), which is usually an RS-232 port. The AUTORUN.SYS program simply triggers the loading. If you are not going to use the four ports on the 850, whether or not you are going to use the printer connected to the 850, make sure the 850 is off when you turn on the computer. If the 850 is on, even if the handler is not loaded, it will reserve some memory for a buffer. This memory is not used or needed if you don't use those four ports.

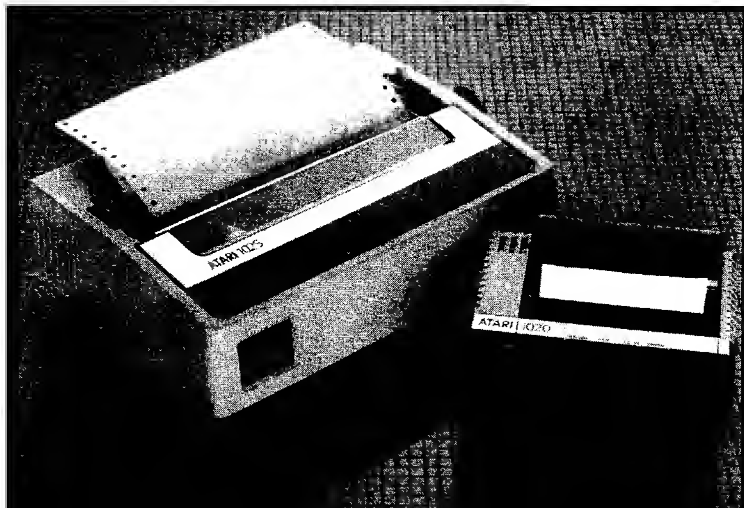
Telecommunications

I recently received a review copy of TeleTari from Don't Ask Computer Software (Lost Angeles, California) and have been using it to play with the local billboards. TeleTari allows you to send and receive disk files, which makes it useful for uploading and downloading programs. It also has a very complete terminal configuration selection, allowing communication to anything your modem can handle.

It does live up to it's "Friendly Terminal" slogan — every one of its functions is adequately documented and relatively easy to use. A few of the nicer features include a buffer that collects everything while you are on line and that can be reviewed on the screen at any time, easily printed, saved on disk or sent to any other peripheral you choose. Also, the buffer doesn't disappear after you save it, so you can, for example, print it out, then save it to disk. You have to tell TeleTari when you want the buffer cleared. Suggested retail price for TeleTari is $39.95.

(continued)

From Here to Atari *(continued)*



ATARI 1025™ 80–COLUMN PRINTER
ATARI 1020™ 40–COLUMN COLOR PRINTER/PLOTTER

Atari 1025 Printer

If you are looking for a printer and don't want to pay the extra money for the 850 interface, you will probably be looking at an Atari 1025. It handles full 9.5 × 11 inch forms, which usually come with perforations so that the sprocket holes rip off leaving you with 8.5 × 11 sheets. It uses fanfold paper only — no single sheets. Suggested retail price for this new printer is $595.

The 1025 has 80 columns per line but doesn't require the 850 interface. It uses a typewriter spool-type ribbon. It prints at 16.5, 10 or 5 characters per inch and 6 or 8 lines per inch. It also has "European" characters.

The 1025 does not support graphics nor does it have proportional print. It is probably worth looking at because the 850 is not required, but if you already have an 850, you may want to look at some other printers. I'll stick with my IDS IP-225 and Epson MX-80.

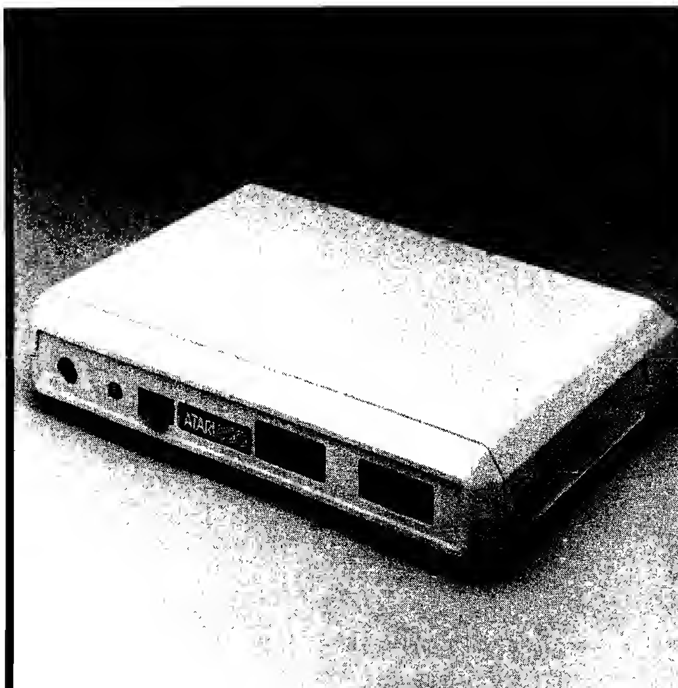
The Atari 1020 printer looks like it will be a little more interesting. It is a four-color "printer" (described like a drum plotter) using colored pens instead of the dot matrix impact printhead. It also connects directly to the serial bus, but uses roll paper only 4" wide. The 1020 suggested retail price is less than $300.

Interfacing Using Controller Jacks

One topic that keeps recurring is peripheral control through the joystick ports. The four jacks on the Atari 400 and 800 computers are full of places to hook up peripherals. I mentioned my IDS IP-225 printer, but since I have only one 850 interface, I have that printer hooked up through controller jacks 3 and 4 on my Atari 400. That particular printer requires only seven data bits. I use the eighth bit available in the jacks for the STROBE line to the printer and hook the printer's ACK line to a joystick trig-

ger input. If CIO makes you think of a bunch of spies, you may want to use PEEK and POKE from BASIC to control the printer. If it means the Central I/O utility, then you may want to write a handler for the printer.

My IDS uses a handler set in place by a BASIC program. The handler occupies the entire top half of page 6 (actually, 126 bytes) and handles OPEN, CLOSE, PUT and initialization, so the "P" device still refers to the controller jacks after you hit SYSTEM RESET. I am preparing an article on this interface to CIO. If I successfully complete it, you will see it here in MICRO. Otherwise, I will continue the description in future columns. I think such a topic is well worth discussing because you can use the same method for attaching almost any peripheral you want through the controller ports — parallel or serial — and have the handler control all handshaking and data transfers one byte at a time.



Promises, Promises

I know I promised a description of the new 80-column board from Austin Franklin Associates, Inc., in this month's column and it is not here. There was a problem in the software at the last minute, which delayed the release beyond the deadline for the column. Look for you next month. MICRO™

Send your letters to Paul Swanson at 97 Jackson St., Cambridge, MA 02140.

Circle No. 22

CoCo Bits

John Steiner

This has been a busy month for me. I am looking forward to attending the Rainbowfest, the first show for the Color Computer, which is being held in Chicago the end of April (a fait accompli by the time you read this). Also, since this is the hardware issue, I have prepared some news and comments on hardware accessories and modifications for CoCo.

New ROMs Issued

Bob Rosen of Spectrum Projects has passed along some information on new ROMs for the Color Computer. According to his release, Bob has seen and tested these ROMs. He finds Color BASIC 1.2 to contain minor changes in the character I/O and interpret integer expression routines. Extended BASIC 1.1 contains a few changes in the graphics routines. Included is a correction for the PCLEAR bug. Both new ROMs are completely compatible with existing software.

As I commented earlier, there is a new 1.1 Disk ROM as well. According to Bob, it has been drastically changed. A DOS command, added to boot alternate operating systems such as OS-9, has necessitated a major revision. The DSKCON routine at $D66C in ROM 1.0 has been moved to $D75F in the 1.1 version. This causes pre-existing software using 1.0 version disk I/O to be incompatible with the new ROM. One other change is the correction of an error in the COPY routine. When encountering an error, COPY can lock up and give an OB error. The new ROM will report only an I/O error and leave everything unchanged.

PCLEAR Bug

For those of you who may not be familiar with the PCLEAR bug, I will describe it more thoroughly. As I commented earlier, Extended BASIC has a bug regarding the PCLEAR command, which is used to reset the number of graphics pages reserved. The problem occurs only when PCLEAR is used from within a large program. When a PCLEAR is issued, it moves the BASIC program in memory to its new location. At this point, it forgets where it was in the program and starts executing from where it thinks it should be. Usually this is in the middle of a line somewhere and a ?SN ERROR message appears. If you type RUN again, the program has already been moved and everything runs normally. The easiest way around this problem is to PCLEAR from immediate mode. I will have more details on working around this bug in a future article.

Lower-Case Adapter

I finally got around to installing my lower-case adapter board from Micro Technical Products into the TDP. I had used it extensively in my other color computer but waited until the warranty expired to install it in my new computer. After I installed the video monitor interface, the lower-case adapter would not fit. I had to get a 40-pin socket to raise the LC board above the other IC adapter socket. The whole thing works well together, and I now have light letters on a dark background, except when using lower case.

The circuit board installs between the 6847 video display generator and its socket and allows you to return to normal display with a dip switch. Another switch is supplied that will switch between the internal 6847 character set or the alternate set included in an external ROM. You may hook up external switches if you want. Due to the large cooling slots on the top of my TDP, I can reach through with a plastic screwdriver and switch the internal switches any time I want — a convenience I never expected.

Dual Cassette Interface

In keeping with this month's hardware theme, I have included a circuit that allows dual cassettes to be hooked up for tape backup purposes. The circuit is simple and contains only a single DPDT relay and a switch. It is powered by a 9-volt battery and is connected to the computer and recorders by the standard 1/16 micro plugs. Figure 1 contains the circuit, which can be built in any mini-box or other suitable enclosure.

To use the interface, take the cable that hooks into the motor switch jack and install it into the micro jack on the interface. Insert the plug marked playback into the recorder motor jack on the machine you wish to use for a source. Insert the other plug into the record-machine motor jack. Remove the plug that goes into the audio input jack on the playback recorder and install it in the auxiliary input on the machine you will use for recording. Insert the source tape into the machine you designate as playback recorder and place a destination tape into the record unit. Put the source recorder into PLAY and the destination unit into RECORD.

To start the ball rolling, CLOAD the first program into tbe machine. When it is loaded, key CSAVE''filename'': CLOAD, but don't press ENTER yet. Press the switch on the interface and hold it in while you press ENTER. The relay will provide power to the record motor only. After the program is saved, the computer motor relay will drop out momentarily, causing the relay in the interface to disengage the record motor and at the same time engage the playback motor circuit. When BASIC encounters the CLOAD after the colon, the internal motor switch engages, causing a new program to be loaded in. A new combined CSAVE : CLOAD command can be issued.

New CoCo Reference Guide

I walked into a Radio Shack store the other day and happened to notice a new reference source for the Color Computer. The book, *TRS-80 Color Computer Quick Reference Guide*, catalog number 26-3194, contains 71

(Continued on page 22)

pages of information on the CoCo, including all BASIC, Extended, and Disk commands, ROM data, character and data codes, and a memory map. In addition, the 6809 instruction set, ZBUG, EDTASM+ commands and error messages are included. Line-printer variables, video and color tests, and computer specifications are also included. At $4.95 this book is quite a bargain.

64K Upgrade

This month, I converted some CoCos from 16K to 64K. The upgrades are easy to do, especially for the E and F boards, and with 6146 ICs at $1.00 a byte, there is little excuse not to upgrade. Remember, though, if you break the seal on the bottom center screw, you will void your Radio Shack warranty. You may want to wait the 90 days before upgrading. Don't tackle this job if cutting foils on circuit boards and fine soldering make you squeamish. To upgrade a D board, replace the 1.0 BASIC ROM with BASIC 1.1. Just remove the old IC and plug in the new one in the same direction. If you have Extended BASIC, you can tell which ROM you have by entering EXEC 41175. The BASIC ROM version number will appear on the screen.

Now the hard part; you must remove all the screws holding the computer to the bottom of the cabinet. In addition, you must remove the power transformer screws. Turn the circuit board over and remove the several metal plugs that hold the shield to the circuit board. Be careful not to lose any of the plastic spacing washers that hold the shield away from the bottom of the board.

Unsolder and remove capacitors C61, C31, C64, C35, C67, C45, C70, and C48. These capacitors lie between the memory chips and are the leftmost of the two capacitors as you face the computer circuit board from the front.

Move the jumper at the right of U10 to the 16K position, and remove completely the jumper plug between U8 and U4. Using an X-acto knife or razor blade, cut the foil that supplies +5 volts to pin 9 of the RAM chips. Then cut the +12-volt line to pin 8 of the RAMs and cut the -12-volt line to pin 1 of the RAMs.

Now you must connect the new RAMs into the existing circuits and add 64K logic circuitry. Use 30-guage wirewrap wire (or equivalent) to make the following connections. The 6146 uses only a single +5-volt supply, which you must connect to pins 1 and 8. Pin 9 of the RAMs must be connected to pin 35 of U10, the SAM chip. Pin 12 of U4 is then connected to pin 16 of U8. If you stop here and install the 4164 ICs, you will have the standard Radio Shack 32K upgrade. To access the full 64K, do the following:

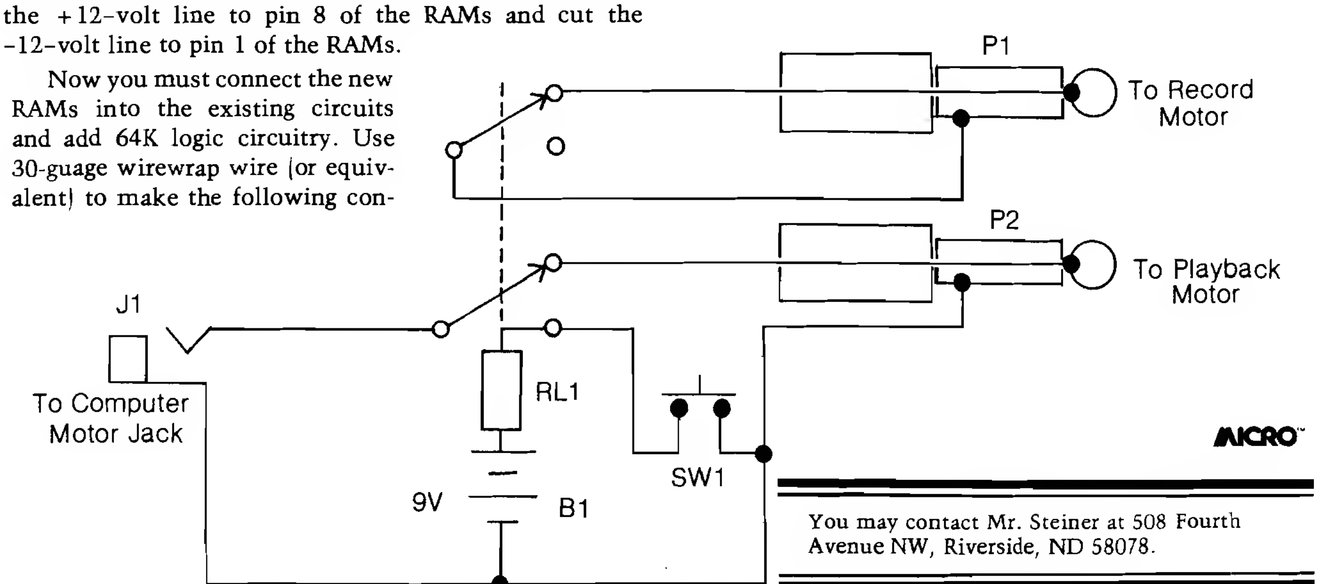
1. Remove U29 and U5 from their sockets and bend pins 4, 5, and 6 of U29 and pin 5 of U11 in the air. Reinsert them and connect pin 6 of U29 to pin 8 of U29.
2. Connect pin 4 of U29 to pin 5 of U11.
3. Connect pin 5 of U29 to TP1.
4. Install 64K chips in sockets U20-U27.
5. Reinstall the bottom shield, and reinstall the circuit board in the cabinet bottom. You may want to power it up and test it before you get it completely put back together.

After completing the upgrade you may find that the computer will power up with a blank screen. Some 6883 SAM chips found in the D boards will not work properly when used with 64K chips. One of the D boards I upgraded required a new 6883, and so you are forwarned. A 6883 is available for about $30.00. Among other sources, Spectrum Projects, 95-15 86 Drive, Woodhaven, NY 11421, provides RAMs with upgrade instructions, SAM chips, and other CoCo parts.

Next month I will detail the 16K-64K upgrade information for E and F boards.

Parts List for Cassette Interface

RL 1	DPDT relay
SW 1	SPST normally open push button switch
J 1	1/16 micro jack
P1, P2	1/16 micro plug
B1	9-volt transistor battery



MICRO

You may contact Mr. Steiner at 508 Fourth Avenue NW, Riverside, ND 58078.

MICRO™

Interface Clinic

by Ralph Tenny

This month we need to complement the multi-bit output scheme developed for RS-232 serial ports.

For any computer that implements the serial port as a software function, Serial In, Serial Out, and Common may be all that is available. On CoCo, the printer software uses Serial In as Busy, but when using a modem (used to allow communication with remote computers via telephone lines), CoCo has a fourth line called CD for Carrier Detect, which the modem uses for "Message coming!"

In previous sessions we have discussed how to program CoCo's Serial In and Serial Out lines, which are port lines on CoCo's PIA I/O (programmable interface adapter). However, the CD line is one of four special lines: CA1, CB1, CA2, and CB2. CA1 and CB1 can be used for input only, and CA2 and CB2 can be used for either input or output. All four of these lines can be used as *interrupt* inputs. An interrupt is a way of gaining the computer's attention to service events not recognized by whatever program is operating. Here's an inexact but useful example: on computers running in BASIC, the BREAK key is monitored by the BASIC interpreter, but a machine-language program will not respond to BREAK unless the program is written to scan the keyboard and watch for the BREAK key. So, if something goes wrong, all you can do is hit the RESET button to regain control. With an interrupt, it would be possible to gain control without using RESET.

Most modern processors have RESET (used for power-up initialization), NMI (non-maskable interrupt) and IRQ (interrupt request). IRQ is recognized by the processor depending on whether the interrupt flag (a special bit in the status register) is set or cleared. NMI will always be recognized at the end of the instruction being processed, and RESET is a shut-down-now-and-start-over signal. We will deal with interrupt programming later.

Only CA1 and CB1 of U4 are available on CoCo for normal programming; CB1 goes to the cartridge slot and CA1 is the CD line on the serial port. You may remember that we access the Data Direction Registers in the PIA by changing Bit 2 of the Control Register from logic 1 to logic zero. A similar form of indirect access is used for the CA and CB lines in the PIA. Figure 1 shows the Control Register contents according to function; as mentioned above, Bit B2 is the familiar switch that allows us to read and write either the DDR or the port at the same address ($FF20 for Port A and $FF22 for Port B). Bits B6 and B7 are interrupt flags, which means that they are set when the input requirements for CA1 and CB1 are met as discussed below.

Figure 1. **Bit assignments and functions of 6821 PIA Control Register.**

B7	B6	B5	B4	B3	B2	B1	B0
IRQA	IRQB	CA2 CONTROL BITS			DDR	CA1 CONTROL	

B6 and B7 are read-only, while the other bits may be read or written. Remember also that the Control Register is accessed when the LSB (least significant bit) of the address is on ($FF21 for CRA and $FF23 for CRB).

For now, we will pass over Bits B3, B4 and B5; I recommend that B5 and B4 be programmed to logic 1 and B3 to logic zero. On CoCo, CA2 of U4 controls the cassette motor and CB2 enables sound output, so this programming leaves those bits inactive.

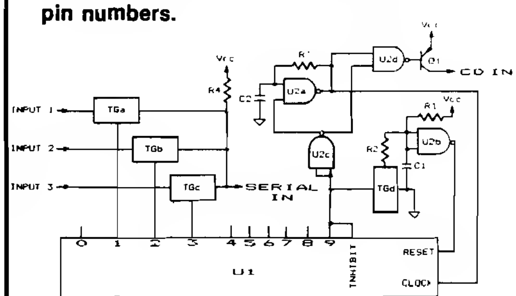
The following mini-program will illustrate how to read CA1 in a noninterrupt mode. Let's review the principle: when CA1 is driven by an external signal, flag B7 in the Control Register is set high. This flag must be reset before another input can be detected, and this is accomplished by reading Port A (read Port B to reset flag B6). Connect 5V to 10 volt source in series with a switch between pins 1 and 3 of the serial port, so that a positive voltage is applied to pin 1 when the switch is closed. Watch what happens in this program if you close the switch after line 30 executes:

```
10 FOR X=1TO1500:NEXT X
20 A=PEEK(65313)
30 PRINT A
40 B=PEEK(65312)
50 GOTO10
```

Line 10 provides a delay that slows down the action for easier observation. Line 20 reads CRA, and line 30 prints the value (probably 52). Line 40 reads Port A, which will reset B7 as it was set, and line 50 starts over. After you close the switch momentarily, line 30 will (probably) print 180. If you convert 52 and 180 to hex numbers, you get $34 and $B4 respectively, which indicates that B7 went high in response to activity on CA1.

Figure 2 is the schematic of an input adapter for a serial port; it can read up to eight bits, converting them to a serial input data stream. Two types of integrated circuits we haven't looked at before are used in this design; the *transmission gate* and the *sequencer*. A transmission gate is implemented in CMOS logic, and its function is like that of a tri-state game — to disconnect one signal from another. Figure 2 shows four such blocks, marked "TG". We must realize that an input signal is passed through to the output only when the control line is high. A sequencer is illustrated by U1 in Figure 2; it is a decimal counter with specially decoded output states "0" to "9". Only one output is active (high) at one time; when U1's RESET line is activated, output 0 goes high. On the first

Figure 2. **Schematic diagram of a multi-bit serial input adapter. Refer to individual device schematics for pin numbers.**



clock pulse, 0 goes low and output 1 goes high. This sequence continues until state 9 "rolls over" to state 0 or until RESET is used again. A sequencer will activate several different func-

tions, one at a time, with the time between events determined by the clock rate for the sequencer. In our circuit, from one to eight TGs will sequentially connect input logic levels to the Serial In line.

Schmitt Triggers were synthesized from inverters in Session 4, but this function is available in CMOS logic as the CD4093. This device, along with the sequencer and transmission gates discussed above, make up our multi-bit input accessory shown in figure 2. Figure 3 shows the timing diagram for the circuit; the following discussion refers to figure 2 and figure 3. U2a is an oscillator that cycles U1 through all its count states. U2b, R1, and C1 hold U1's RESET low (inactive) for most of an operational cycle. TGd and R2 pull down on R1 and C1, slowly discharging C1 to U2b's threshold, causing it to reset U1. This reset operation is started when output 9 goes high at the end of an operational cycle.

Let's pick up the operation when state 9 comes true. Only TGd is turned on, so the Serial In line is pulled high by R3 and TGd has begun to discharge U2d are inhibited by U2c, stopping the clock and input on CD. When C1 is clock and input on CD. When C1 is discharged far enough, U2b activates RESET and state 9 changes to state 0. U2a and U2d are released, allowing a normal cycle to start. The time constant R1/C1 was deliberately chosen to give a long "off" time for the CD input, so that a program can more easily determine when the cyle starts. Once U2d is enabled again, CD is pulled high (a logic zero at the CD pin on U4), once for each clock cycle.

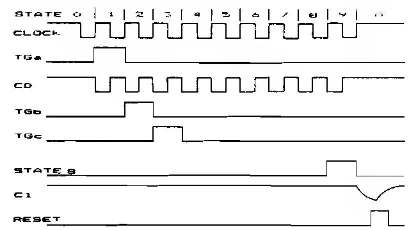
A normal operational cycle will begin with a pulse for 0 state when no input is expected. There will then be an input on CD and a logic level input on Serial In for each successive state through state 8, then the cycle terminates when state 9 happens again.

Programming to read this input accessory should accomplish the following sequence:

1. Initialize U4's Control Register A to $34.
2. Read Port A to reset B7.
3. Monitor B7 in a loop, waiting for it to go high.
4. When B7 goes high, check for loop time, and identify the long cycle. Read Port A to reset B7.
5. Catch the next pulse (state 0), then read Bit 0 of Port B for eight successive pulses, resetting B7 each time.
6. Assemble the eight input bits into a data byte and take whatever action is dictated by data data.

Of course, either assembly language or BASIC can be used for this program, but the relative time constants may need to be adjusted for faster operation if assembly language is used. Next time: programs solving the above problem, contrasting BASIC, assembly language and interrupt programming.

Figure 3. Timing diagram for the serial input adapter.



Parts List for the Input Accessory:

U1 - CD4017, Radio Shack 276-2417
U2 - CD4093, R/S 276-2493
TG - CD4066, R/S 276-2466
C1 - .22 uF, 16 volt (or more) capacitor
C2 - .001 uF, 16 volt capacitor
R1 - 22k ohm, ¼ watt resistor
R2 - 8.2k ohm, ¼ watt resistor
R3 - 1 megohm, ¼ watt resistor
R4 - 4.7k ohm, ¼ watt resistor
Q1 - 2N3906 PNP transistor, R/S 276-1604
VCC - +5v to +10v (Battery OK)

Please forward questions and suggestions for discussion topics to Mr. Tenny at P.O. Box 545, Richardson, TX 75080.

Listing 1

```
                  * This program will calibrate a Serial Port Adapter
                  *
                  * Equates
           FF20   PORT    EQU    $FF20     SERIAL IN port
           FF21   CTLR    EQU    $FF21     Control register
           0020   COUNT   EQU    $20       Count register
1000                      ORG    $1000
1000 86   34      START   LDA    #$34      Init control register
1002 B7   FF21            STA    CTLR
1005 8E   0014            LDX    #20       Set index
1008 B6   FF20            LDA    PORT      Clear IRQA
100B 0F   20      STRT1   CLR    COUNT     Zero counter
100D F6   FF21    IN      LDB    CTLR      Test for IRQA set
1010 0C   20              INC    COUNT     Record operation
1012 C4   80              ANDB   #$80      Mask to MSB
1014 27   F7              BEQ    IN        Not set, Try again
1016 B6   FF20            LDA    PORT      Otherwise, Clear IRQA
1019 96   20              LDA    COUNT     and keep a record
101B A7   89 1024         STA    BUFR,X
101F 30   1F              LEAX   -1,X      Decrement index
1021 26   E8              BNE    STRT1     Loop until Index = 0
1023 39           EXIT    RTS              Then quit
1024              BUFR    RMB    20        List of counts
                          END    START


                  * This program will input from a Serial Port Adapter
                  *
                  * Equates
           FF20   PORTA   EQU    $FF20     SERIAL IN port
           FF21   CTLR    EQU    $FF21     Control register
           FF22   PORTB   EQU    $FF22
           0020   COUNT   EQU    $20       Count register
1000                      ORG    $1000
1000 86   34      START   LDA    #$34      Init control register
1002 B7   FF21            STA    CTLR
1005 8E   0000            LDX    #0        Set index
1008 B6   FF20            LDA    PORTA     Clear IRQA
100B 0F   20      STRT1   CLR    COUNT     Zero counter
100D F6   FF21    IN      LDB    CTLR      Test for IRQA set
1010 0C   20              INC    COUNT     Record operation
1012 C4   80              ANDB   #$80      Mask to MSB
1014 27   F7              BEQ    IN        Not set, Try again
1016 B6   FF20            LDA    PORTA     Otherwise, Clear IRQA
1019 96   20              LDA    COUNT     Test for long cycle
101B 81   25              CMPA   #$25
101D 23   EC              BLS    STRT1     Loop until long cycle
101F 0F   20      STRT2   CLR    COUNT     New set of numbers
1021 F6   FF21    IN2     LDB    CTLR      Test for IRQA again
1024 0C   20              INC    COUNT     Count operations
1026 C4   80              ANDB   #$80      Mask to MSB
1028 27   F7              BEQ    IN2       Loop until new edge
102A B6   FF20            LDA    PORTA     and reset IRQA
102D B6   FF22            LDA    PORTB     Read RS232 line
1030 A7   89 104D         STA    BUFR,X
1034 96   20              LDA    COUNT     Test count
1036 81   25              CMPA   #$25
1038 22   04              BHI    BUILD     Assemble input word
103A 30   01              LEAX   1,X       Increment index
103C 20   E1              BRA    STRT2     Loop until long cycle
103E 86   0B      BUILD   LDA    #11       Set a counter
1040 64   89 104D SHIFT   LSR    BUFR,X    Shift bit into carry
1044 79   1059            ROL    WORD      Shift bit into word
1047 30   1F              LEAX   -1,X      Step back through buffer
1049 4A                   DECA             Count bits
104A 26   F4              BNE    SHIFT     Loop until 8 bits recovered
104C 39           EXIT    RTS              then quit
104D              BUFR    RMB    12        List of counts
1059 00           WORD    FCB    0
                          END    START
```

Buyer's Guide

by Phil Daley and Loren Wright

The next nine pages offer information that will help you choose a system to suit your needs. A detailed chart, beginning on page 28, provides comparisons of important features found on the ten systems shown and described here.

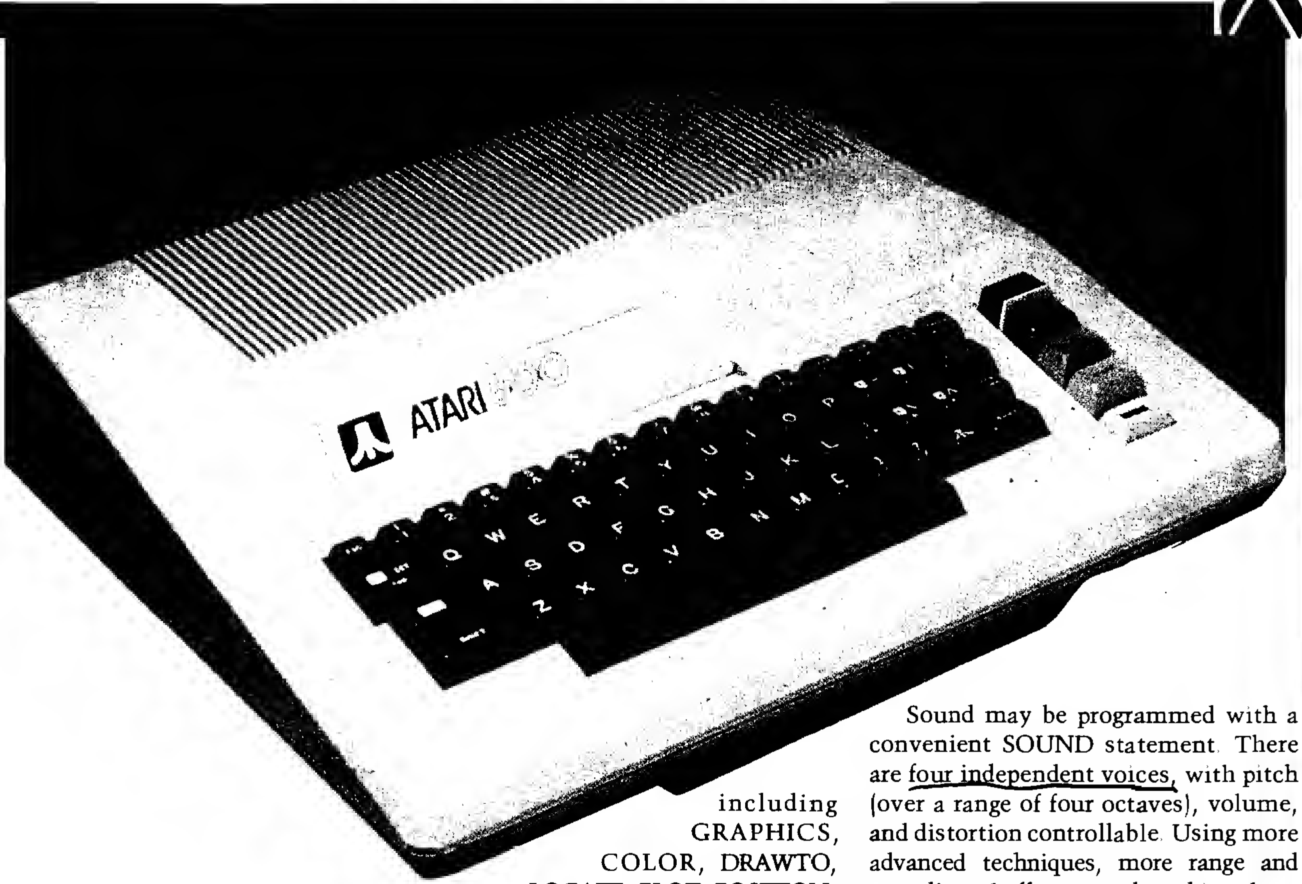
Atari 800

The Atari 800 has been the flagship of the Atari computer line since 1981. It includes a full-stroke keyboard and 48K RAM. An Atari BASIC cartridge is optional. At press time, the Atari 800 listed for $679, but was available at considerable discount. Atari computers use their own serial bus for connection to peripherals, such as disk drives, printers, modem, cassette machine, and the 850 interface module. In addition, there are four controller ports, which may be used for joysticks, paddles, lightpens, and other parallel or A-to-D interfacing applications. An RF modulator is built-in, with a cable included for hook-up to a TV splitter. The Atari 800 also has a connector for output to a video monitor.

With 16 colors available in 16 different luminances, the Atari has an effective pallette of 256 different colors. There are 12 different graphics modes available; three are text and nine are graphics. In addition, other modes are available in machine language or with BASIC POKE statements, making a total of 17. The graphics modes range from a one-color (two-luminance)

Atari 400

The Atari 400 is essentially the same machine as the 800 in a less expensive package. The keyboard is a flat membrane type. (There is audio feedback on as each key is pressed. Also, a replacement full-stroke keyboard is available from InHome Products and others.) There is no video monitor output and no right cartridge slot. Not much software was written for right cartridges, and the new 1200XL doesn't include the slot either. The Atari 400 comes with 16K, but this may be upgraded by a second-source board. Generally, the same software runs in both the 400 and 800. The list price at press time is $199, but this is heavily discounted.



40 x 24 character mode, through an 80 x 192 16-color mode, to a one-color 320 x 192 high-resolution mode. Options may be exercised to prevent the default split-screen mode or to prevent clearing the screen. Atari BASIC has a full set of convenient graphics keywords, including GRAPHICS, COLOR, DRAWTO, LOCATE, PLOT, POSITION, PUT/GET, and SETCOLOR. Movable object blocks, called players and missiles, can be programmed 256 bytes high and 8 bits (players) or 2 bits (missiles) wide. The system can maintain four players and four missiles or, alternately, five players.

Sound may be programmed with a convenient SOUND statement. There are four independent voices, with pitch (over a range of four octaves), volume, and distortion controllable. Using more advanced techniques, more range and complicated effects may be achieved.

When the BASIC language cartridge is removed, the memory it normally occupies becomes available for alternate languages, such as FORTH, Pascal, LOGO, and Microsoft BASIC, or for machine-language programs, such as sophisticated games.

ATARI 1200XL HOME COMPUTER™

Atari 1200XL

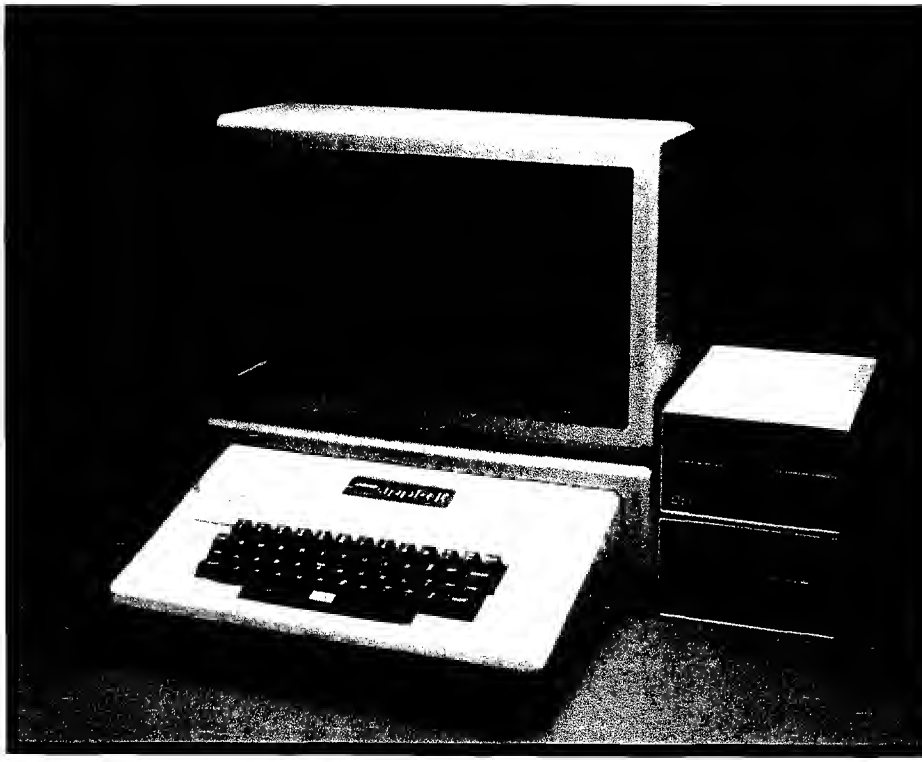
The 1200XL, Atari's new machine, is essentially similar to the 400 and 800. It has a more streamlined appearance. The keyboard has been greatly enhanced by the relocation of a number of keys and by the addition of programmable function keys. Also, cursor movements are accomplished with single keys. The 1200XL has an international character set available. There is only one cartridge slot, but there is very little existing software that requires the other slot. There are only two controller ports.

The operating system has been rewritten, such that four more graphics modes are conveniently programmable from BASIC. (These modes are present in the 400 and 800, but they can only be accessed using an advanced technique called display list programming.) Some existing software for the 400 and 800 will not run on the 1200XL as a result of the operating system rewrite.

SYSTEM	Manufacturer's Name	Manufacturer's Address	First Produced	Microprocessor	Memory: Standard	Memory: Expandable to	Price	Keyboard: Type	Keyboard: U/L Case	Keyboard: # Keys	Keyboard: # Key Codes Gen.	Keyboard: Character Set	Display: TV or Monitor	Display: RF Modulator	Display: L/C Descenders	Display: # Char Displayed	Display: #Characters Per Line	Display: Lines Per Screen	Graphics: Characters: Grid Colors	Graphics: Low-Res: Grid Colors	Graphics: High-Res Grid Colors	Cartridge	Power Supply	I/O Ports-Types RF & Tape Assumed	Controller Ports	Sounds Produced: Tones	Sounds Produced: #Voices	Sounds Produced: Range	Sounds Produced: Volume Control/Tempo
Apple II +	Apple Computer Company, Inc.	10260 Bandley Dr. Cupertino, CA [illegible]	1978	[illegible]	[illegible]	[illegible]	[illegible]	Standard	[illegible]	52	98	ASCII	Monitor	[illegible]	N/A	[illegible]	[illegible]	24	None	40 × 48 16	280 × 192 6	No	Built-in	8 Expansion Slots	1 Game Port	Simple	None	N/A	N/A/N/A
Apple IIe	Apple Computer Company, Inc,	10260 Bandley Dr. Cupertino, CA 95014	1983	6502A	16K ROM 64K RAM	128K	$1395	Standard	Yes	63	128	ASCII	Monitor	Extra	Yes	256	80	24	None	40 × 48 16	280 × 192 6	No	Built-in	1 RS 232 Expansion Slots	1 Game Port	Simple	None	N/A	N/A/N/A
Atari 400	Atari Home Computer Div.	P.O. Box 427 Sunnyvale, CA 94086	1979	1.79 MHZ 6502B	16K	48K	$199	Flat Membrane	Yes	61	256	ATASCII	TV	Built-in	1 or 2 Scan	256	40	24	128 Colors Available	80 × 192 9 Colors	320 × 192 2 Colors	Yes (1)	In-Line	Serial	4	Complex	4	4 Octaves +	0-7/-
Atari 800	Atari Home Computer Div.	P.O. Box 427 Sunnyvale, CA 94086	1979	1.79 MHZ 6502B	48K	48K	$679	Standard	Yes	61	256	ATASCII	TV/MON	Built-in	1 or 2 Scan	256	40	24	128 Colors Available	80 × 192 9 Colors	320 × 192 2 Colors	2 Cartridge	In-Line	Serial	4	Complex	4	4 Octaves +	0-7/-
Atari 1200	Atari Home Computer Div.	P.O. Box 427 Sunnyvale, CA 94086	1983	6502B	64K	—	$899	Standard	Yes	66	256	ATASCII	TV	Built-in	1 or 2 Scan	256	40	24	128 Colors Available	80 × 192 9 Colors	320 × 192 2 Colors	1 Cartridge	In-Line	Serial	2	Complex	4	4 Octaves +	0-7/-
Commodore 64	Commodore Business Machines	1200 Wilson Dr. West Chester, PA 19380	1982	6510	64K	—	$495	Standard	Yes	65	155	PET ASCII	Either	Built-in	1 Scan line	256	40	25	16	16 Possible 200 × 160 4 at once	16 Possible 320 × 200 2 At Once	Yes	In-line	IEEE Serial Parallel Exp. Cartridge	2	Complex	3 Music 1 Noise	9 Octaves	0-15/-
VIC-20	Commodore Business Machines	1200 Wilson Drive West Chester, PA 19380	1981	6502	5K RAM	29K	$199.95	Standard	Yes	65	155	PET ASCII	Either	Included	1 Scan line	256	22	23	8	16 Possible 184 × 88 4 At Once	16 Possible 184 × 176 2 At Once	Yes	In-line	IEEE Serial Parallel Exp. Cartridge	1	Simple	3 Music 1 Noise	5 Octave	0-15/-
PET 4032	Commodore Business Machines	1200 Wilson Dr. West Chester, PA 19380	1977	6502	32K	—	$795	Standard	Yes	74	139	PET ASCII	Built-in Mon.	N/A	1 Scan line	256	40	25	40 × 25 B + W	N/A	N/A	No	Built-in	IEEE-488 Parallel Expansion	No	No	N/A	N/A	—
TRS-80C	Tandy Corporation	One Tandy Center Fort Worth TX 76102	1981	6809E	8K ROM 16K RAM	16K ROM 32/64K RAM	$299.95	Calculator	Yes	53	98	ASCII	TV	Built-in	N/A	112	32	16	64 × 32 9	128 × 96 4	256 × 192 2	Yes	Built-in	RS 232 Cartridge	2 Game Ports	Simple	1	5 Octaves	1-31/1-255
TI 99/4A	Texas Instruments Inc.	P.O. Box 73 Lubbock, TX 79408	1981	TI 9900	16K RAM 26K ROM	48K RAM	$150.00	Standard	Yes	48	128	ASCII	Monitor	Extra	N/A	97	32 or 40	24	None	32 × 24 16	256 × 192 2	Yes	In-Line	RS 232 (Extra)	1 Game Port	Complex	3	5 Octaves	N/A/N/A

	Mass Storage Devices: Tape Recorders		Mass Storage Devices: Disk Drives						Game Paddles Joysticks				BASIC					Printer			
SYSTEM	#of Units	Price	#of Drives	Capacity Per Drive	Price	Disk Operating Sys	Expansion Memory Size / Price	Modem Type / Price	Type	Compatibility	Proportional Switch	Price	Size	Additional Available/Size	Price	Resident M/L Mon.	Price if not built in	Name	Type of Interface	Special Cables	Price
Apple II +	1	Standard-$50	6	143K	1st $545 Add $395	RAM	16K $195	N/A	Paddles	None	Proportional	$34.95	8K	Integer BASIC 6K	Included w/DOS	Yes	—	Silentype	Non-Standard	Yes	$395
Apple IIe	1	Standard-$50	6	143K	1st $545 Add $395	RAM	64K $289	N/A	Paddles	None	Proportional	$34.95	8K	Integer BASIC 6K	Included w/DOS	Yes	—	Silentype	Non-Standard	Yes	$395
Atari 400	1	Special $99.95	4	88K	$599.95	RAM	32K or 48K $105	835 Telelink II $279.95	Joystick (4) Paddles (8)	Commodore Atari Games	Switch	$20/pair	8K	BASIC A + Microsoft	—	No	—	1025	Serial	Included	$549
Atari 800	1	Special $99.95	4	88K	$599.95	RAM	32K or 48K $105	835 Telelink II $279.95	Joystick (4) Paddles (8)	Commodore Atari Games	Switch	$20/pair	8K	BASIC A + Microsoft	—	No	—	1025	Serial	Included	$549
Atari 1200	1	Special $99.95	4	88K	$599.95	RAM	32K or 48K $105	835 Telelink II $279.95	Joystick (2) Paddles (4)	Commodore Atari Games	Switch	$20/pair	8K	BASIC A + Microsoft	—	No	—	1025	Serial	Included	$549
Commodore 64	1	Special $80	10	176K	$400	ROM	N/A	Vic Modem $110	Joystick (2) Paddles (2)	Atari Games and Computers VIC	Switch	$20/pair	12K	BASIC 4 4K	$100	No	$60	VIC 1525	IEEE-Serial	Included	$395
VIC-20	1	Special $80	10	176K	$400	ROM	3K, 8K, 16K, 24K $60, $100 etc.	Vic Modem $110	Joystick (1) Paddles (2)	Atari Games and Computers Commodore 64	Switch	$20/pair	12K	BASIC 4 4K	$100	No	$60	VIC 1525	IEEE-Serial	Included	$395
PET 4032	2	Spacial $80	10	178K	$499	ROM	—	Anchor $125	No	—	N/A	—	18K	—	—	Built-in		CBM 4022	IEEE-488	P-I Cabla	$499
TRS-80C	1	Special $59.95	4	156K	1st $549.00 Add. $399.00	ROM	32/64K $149.00	AC3 $149-Acoustic DM1 $149-Direct DM2 $249-Direct	Pair Joysticks	None	Proportional	$34.95	8K	Extended BASIC 8K	$99.00	No	$49.95	DMP-100	RS-232	Yes	$399
TI 99/4A	2	$50 Special Cable	3	92K	$749.90 Add $499.95	ROM	32K $299.95	The Telephone Coupler $199.95	Remote Controller	None	Switch	$34.95	—	Extended BASIC	$99.95	N/A	—	Impact	RS 232	No	$750.00

Apple II, Apple II +



The Apple computer is probably the most adaptable computer on the market, and has had the most software written for it of any computer in the world. Each hardware upgrade has maintained compatibility with previous models and each software upgrade has provided conversion programs and upward compatibility. This relatively constant background and modest cost has produced the largest quantity of software available for a single computer. Many schools use Apples as the main computer learning tool and a great quantity of educational software has followed.

The Apple II hardware architecture was designed to make interfacing the computer to external devices easy, and the multiplicity of add-on hardware attests to the success of this strategy. Currently available microprocessor additions include Z-80, 8080, 6809, and 68000 cards with a 16032 card in the works.

and

Almost every computer language in the world can be run on the Apple, including a new Pascal-like language, Modula-2, by Nicholas Wirth. The Applesoft BASIC language includes low- and high-resolution plotting commands that simplify creating graphics pictures, and there are many machine-language programs available for creating animation and detailed graphics designs. The biggest criticism levelled at the Apple was the lack of lower case, which has been corrected on the IIe, and additional hardware can be bought to upgrade older Apples for lower-case display and entry.

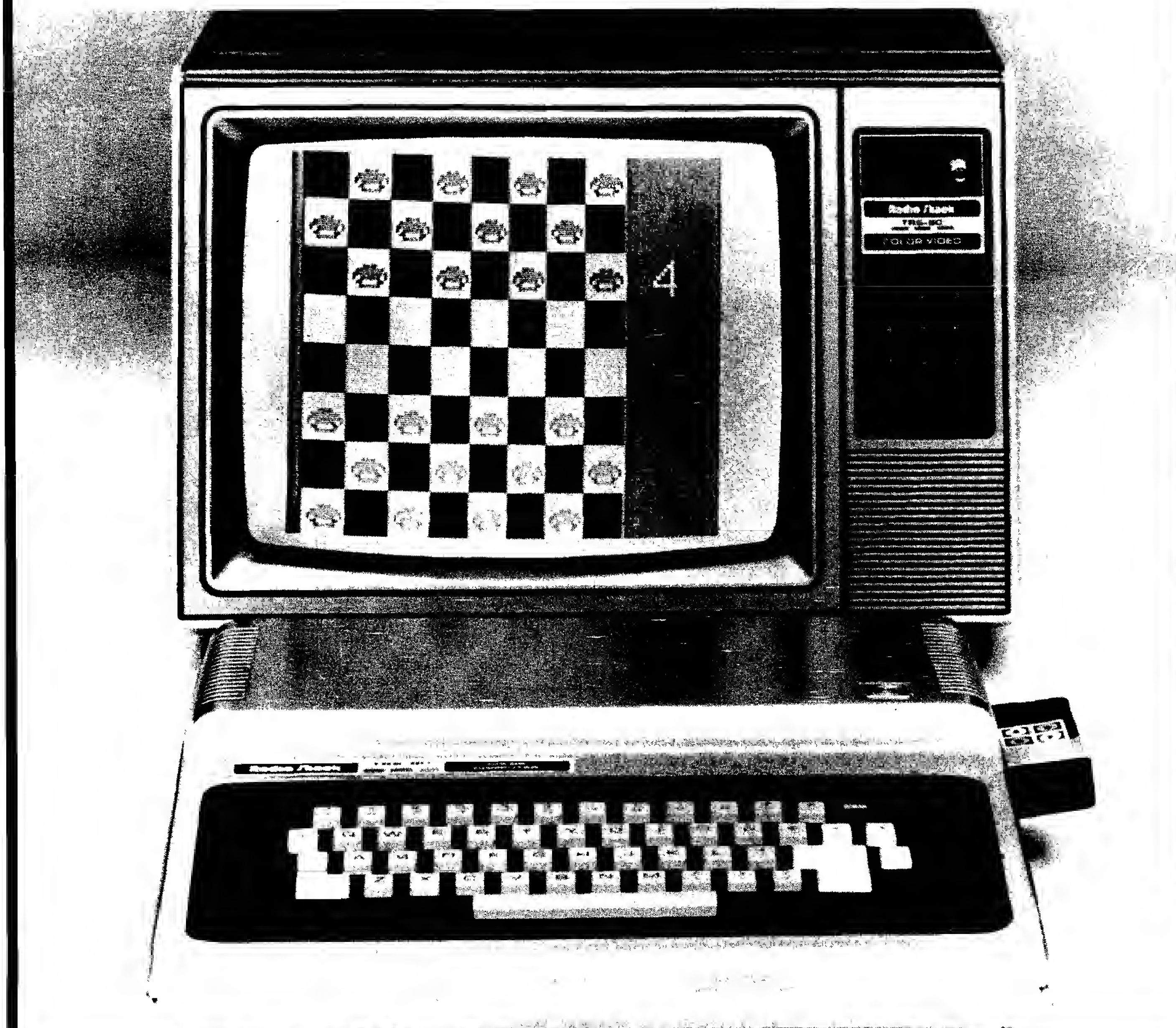
Apple IIe

TRS-80 Color Computer

The CoCo, as it is known, has tremendous potential to become the most versatile of the low-cost computers. Expanded to 32K RAM, a Color disk system can be added (156K per 5" double-density disk), and with minor or no alterations, depending on the manufacturer, it can access the full 64K RAM present and run the operating systems for the standard 6809 micros, Flex, and OS-9. With Extended BASIC, it has the largest and most versatile BASIC among all the low-cost micros.

The CoCo has five graphics modes with special commands for drawing lines and circles, copying pictures from one screen to another, and filling specified areas with a specified color. The string functions include commands to search a string for a specified string, make a string of specified length out of a specified character, convert to and from hexadecimal, and replace one portion of a string with another. For formatted output, the BASIC includes PRINT USING, which has decimal point alignment, comma insert, dollar sign lead, asterisk lead, floating dollar sign, sign before or after, exponential format, and leading spaces. The music command includes note names or numbers (including sharps and flats), octave, volume, tempo, note length, and pause.

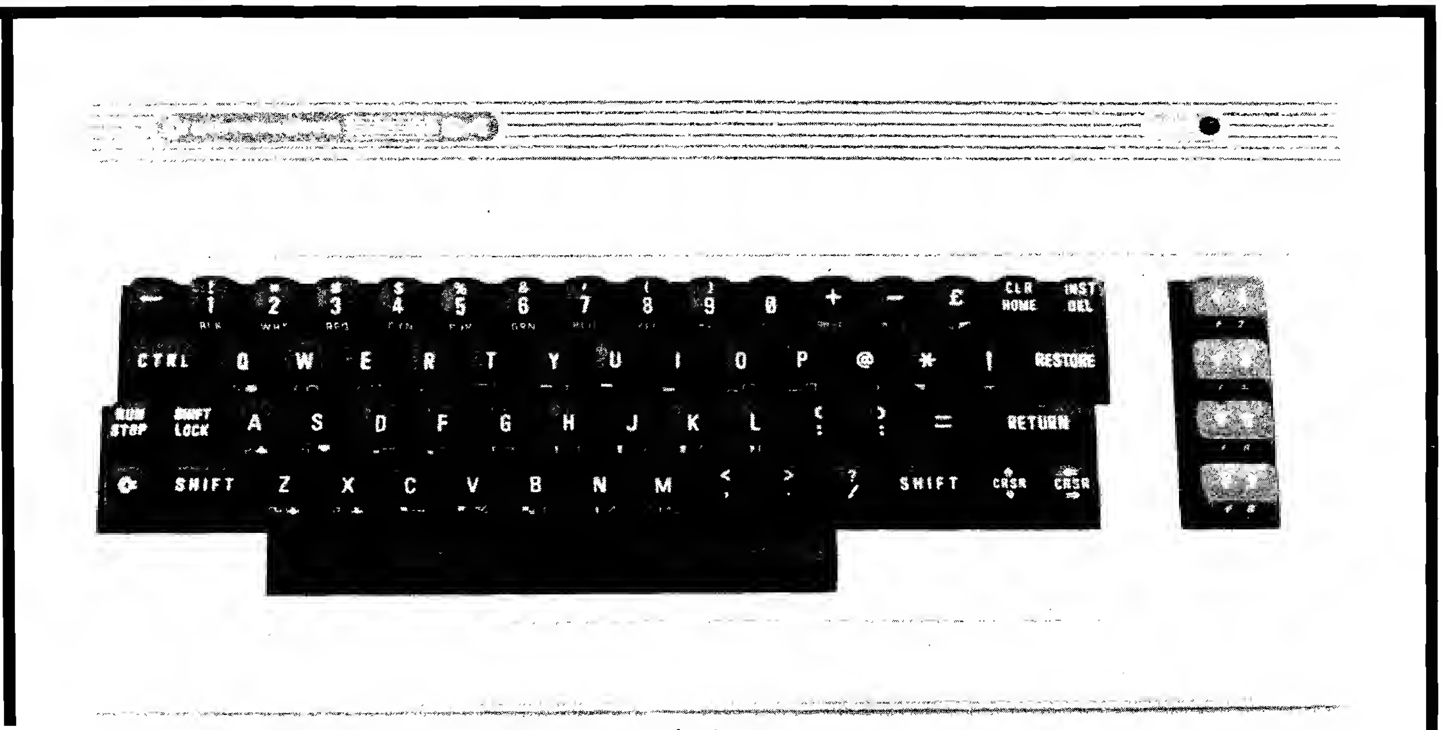
Even though Radio Shack is not giving as much support to the product as they could, several CoCo-only magazines have sprung up, many third-party manufacturers have started producing both software and hardware, and a good deal of grass-root support has sprung up as attested to by the showing at the Rainbowfest in Chicago (see CoCo Bits).



VIC-20

The lowest priced computer in Commodore's line at press time was listed at $199.95, but discounted as low as $88. The VIC-20 features a color display of 23 rows of 22 characters, 5K of RAM (expandable to 29K), and a full-featured BASIC-in-ROM. Two character sets are easily available from the keyboard. One contains capital letters and an extensive assortment of graphic characters. The other replaces the capital letters with lower-case letters, and replaces some of the graphic characters with upper-case letters. Eight colors are available for characters and the border; 16 are available for the screen and auxiliary register. The pointer to the character ROM may be changed to point to RAM, where you can store your own character definitions. Additional capabilities include double-height characters and multicolor-mode characters. The latter allows four colors to be used at once in a character. Limited high-resolution graphics are possible.

The VIC has three voices for simple music, covering a five-octave range, and another voice for noise. The Super-Expander cartridge adds 3K of RAM and convenient commands for graphics and sound. Additional upgrades include 3K, 8K, 16K and 24K RAM cartridges, the inexpensive 1541 disk drive that stores 176K per 5'' diskette, and the 1525 dot-matrix printer. At $110 list, the VIC-Modem is one of the least expensive available. An IEEE-488 adaptor cartridge, available from a number of non-CBM manufacturers, allows use of Commodore dual disk drives, hard disk units, letter-quality printers, and a large variety of scientific instruments manufactured by Hewlett-Packard, Tektronix, Fluke, and others.

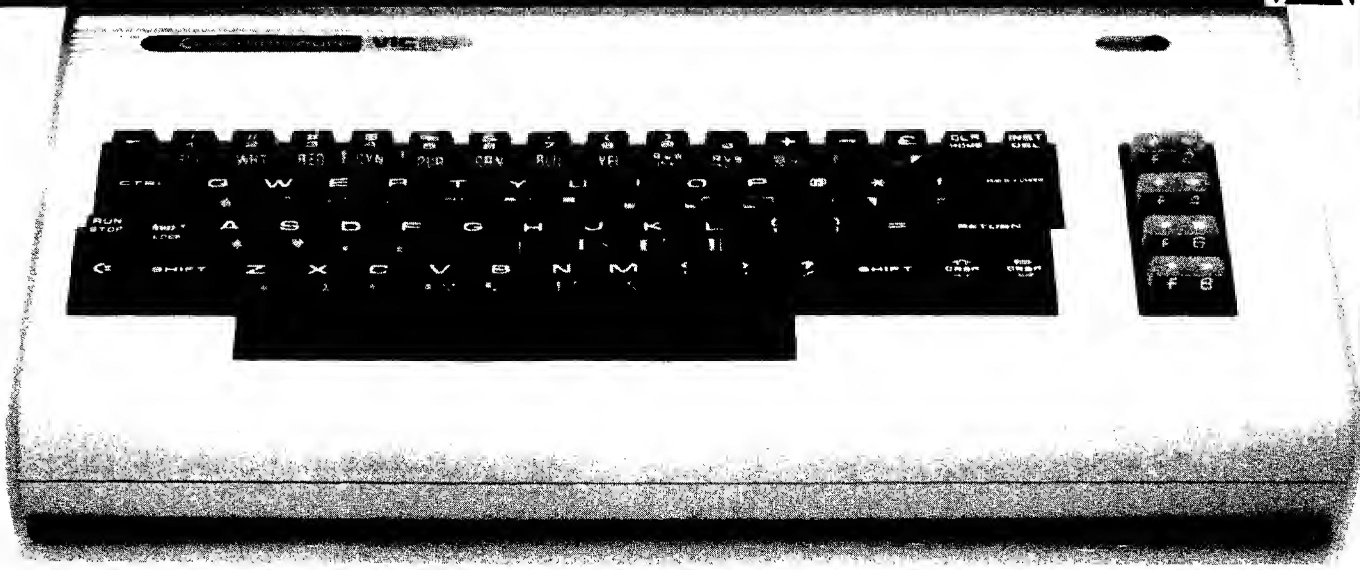


Commodore 64

The Commodore 64 now lists for about $499.95, but is available at considerable discount. The C64 includes 64K, a 40×25-character color display, and BASIC-in-ROM essentially indentical to that of the VIC-20. The 6510 microprocessor allows bank switching of color memory, character-generator ROM, and I/O so that nearly 39K is available for BASIC programs. For other applications, such as alternate languages and machine-language programs, an additional 8K to 16K can be freed. Character graphics are identical to those of the VIC-20, except that eight additional character and border colors are available. An additional controller port (for a total of two) and a built-in RF modulator distinguish the C64 from the VIC externally.

Eight movable object blocks, called sprites, may be programmed. These are maintained by the VIC II CRT controller chip. Each sprite may be 24×21 pixels in size. The color and X,Y-position are specified and the sprite is displayed at that position on the screen. Features include multicolor mode, X-expand, Y-expand, sprite collision detection, and background collision detection.

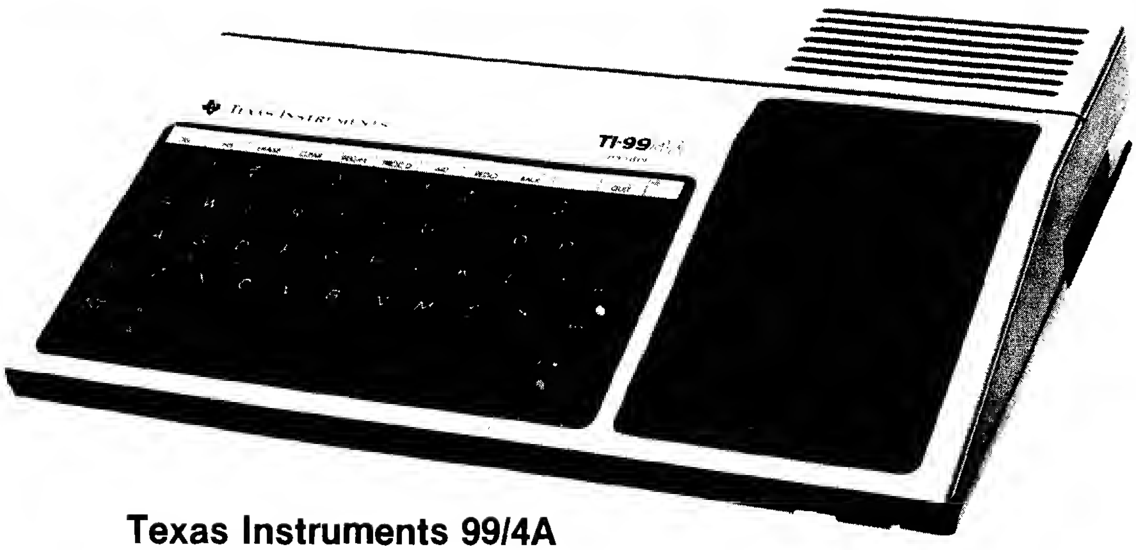
The SID (sound interface device) chip allows three programmable music voices. The frequency is selectable over a range of nine octaves, with triangle, sawtooth, variable-pulse-width rectangular, and noise waveforms available. Each voice has a programmable envelope generator, where attack, decay, sustain, and release may be programmed. In addition, high-pass, low-pass, and band-pass filters may be selected, and synchronization and ring modulation between two voices can be programmed. The audio signal is compatible with high-fidelity amplifiers, and the quality approaches that of dedicated synthesizers.



PET 4032

The PET was the original Commodore computer introduced in 1977. In today's advanced configuration, the 4032 includes a full-sized keyboard with separate numeric keypad. The black-and-white display is directed to the built-in 12-inch green-phosphor monitor. Interfaces include two cassette ports, a parallel port, expansion port, and IEEE-488 port. The 4032 has 32K RAM and BASIC 4, occupying 18K ROM. There are two character sets available, one with upper case and graphics characters, and the other with lower case and upper case. The graphics character set is one of the most complete in a home computer. BASIC programs are generally compatible among PET, VIC-20, and Commodore 64.

Peripherals available from Commodore for the IEEE-488 bus include several printers; .5 Mbyte, 1 Mbyte, and 2 Mbyte dual floppy disk drives; a single floppy drive; and 5 Mbyte and 7.5 Mbyte hard disk units. Because the IEEE-488 is an industry standard, many sophisticated scientific instruments from Hewlett-Packard, Tektronix, Fluke, and others can be used with the PET.



Texas Instruments 99/4A

The TI is a home computer, which is also aimed at the educational market. It has impressive built-in hardware capabilities including a 16-bit microprocessor, the only low-cost computer to have one, and graphic sprites. The only languages available are BASIC, Pascal, Pilot, and LOGO. Almost the only software available is from Texas Instruments, and most of it is on cartridges. TI makes a speech synthesizer to allow the terminal to talk to you. While Texas Instruments is offering good support for the product, almost no one else in the computer marketplace is making any software or hardware for this computer.

MICRO

PrintMate™ 150

THE MOST ADVANCED PRINTER IN ITS CLASS.

Easy-to-use. Fast. Excellent print quality.

These are the features every pc and microsystem user wants. These are the same features we build into every PrintMate 150. And there's one more thing... value. No other printer offers more versatility for the price. At $995, the wide-carriage PrintMate 150 is an exceptional value.

HIGH SYSTEM THRUPUT
Print at 150 cps. Advanced logic-seeking impact printing. Accelerated print head slew rate. High-speed paper advance.

PRINT CAPABILITIES
10, 12, 15 or 17 characters per inch. High-density, high-speed correspondence printing, plus double-wide for emphasis.

SoftSwitch™ KEYPAD
Establish, change or display the operating mode from the front of the printer.

EXPANDABLE PRINT BUFFER
Increase performance with buffers that take you from 2K to 68K to provide high-speed interleaved printing with computing.

EASY WITH PAPER
Three paper paths—front, back or bottom—make the PrintMate 150 one very easy-to-use machine.

DOWNLINE LOADABLE FONTS
Custom character sets may be downloaded to a PrintMate 150 with a 4K or larger buffer.

PRINT WITH STYLE
No other printer offers more graphics support for the price. It's true. Our exclusive AP-PAK™, available for most popular computers, lets you print in dozens of stylized fonts, in characters up to ⅝" high. Got a graph on the screen? Need a custom font or logo? Do it with an AP-PAK.

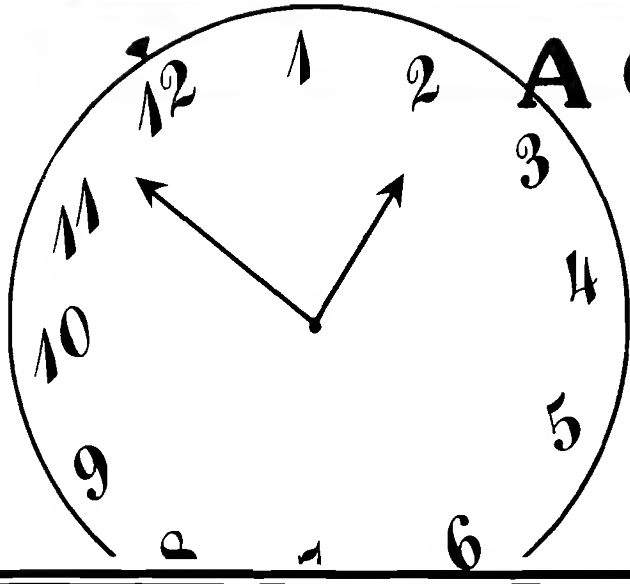
OUR BOTTOM LINE
Superior performance starts at $995 on a PrintMate 150. We wouldn't print that if we didn't mean it. Call or write us today.
1-800-821-8848

OEM's: Ask us about custom AP-PAKs and low-cost private labeling.

The Printer People
mpi

Micro Peripherals, Inc.
4426 South Century Drive
Salt Lake City, UT 84107

Circle No. 27



A Clock Interrupt for Your Apple

by Charles Putney

This hardware modification and interrupt-driver program allows use of 1/60-second interrupts on the Apple. The driver demonstrates how interrupts are serviced and implements a time-of-day clock with a type-ahead buffer.

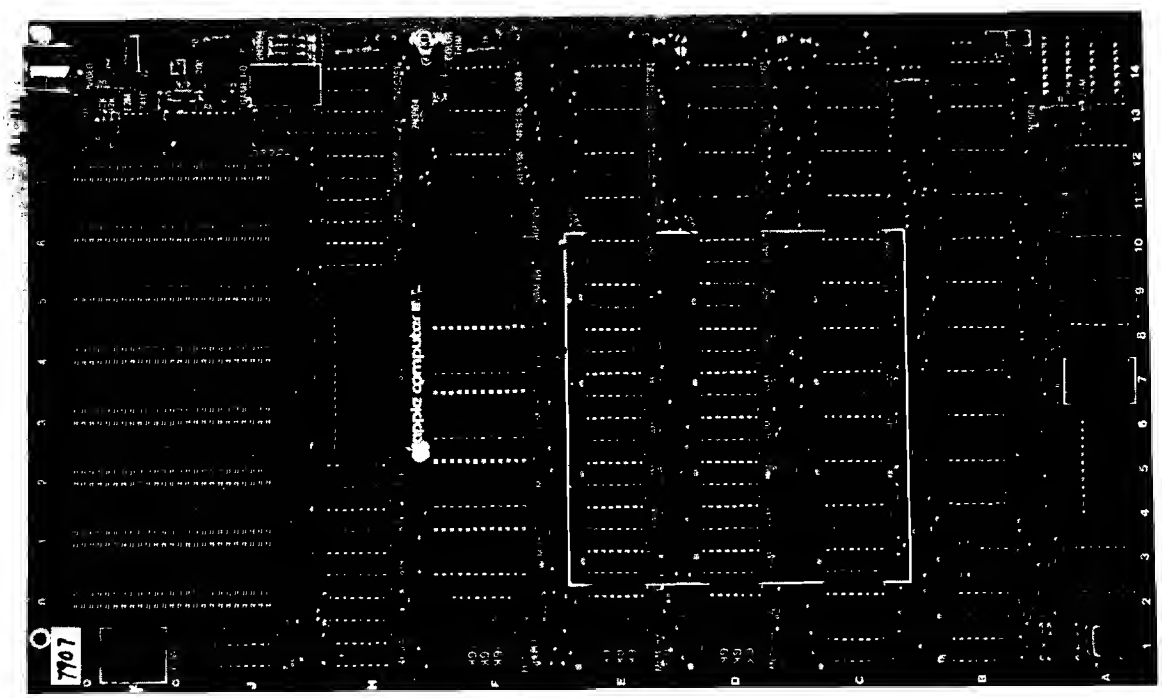
Clock Interrupt
requires: Apple II w/16K RAM
A one wire modification

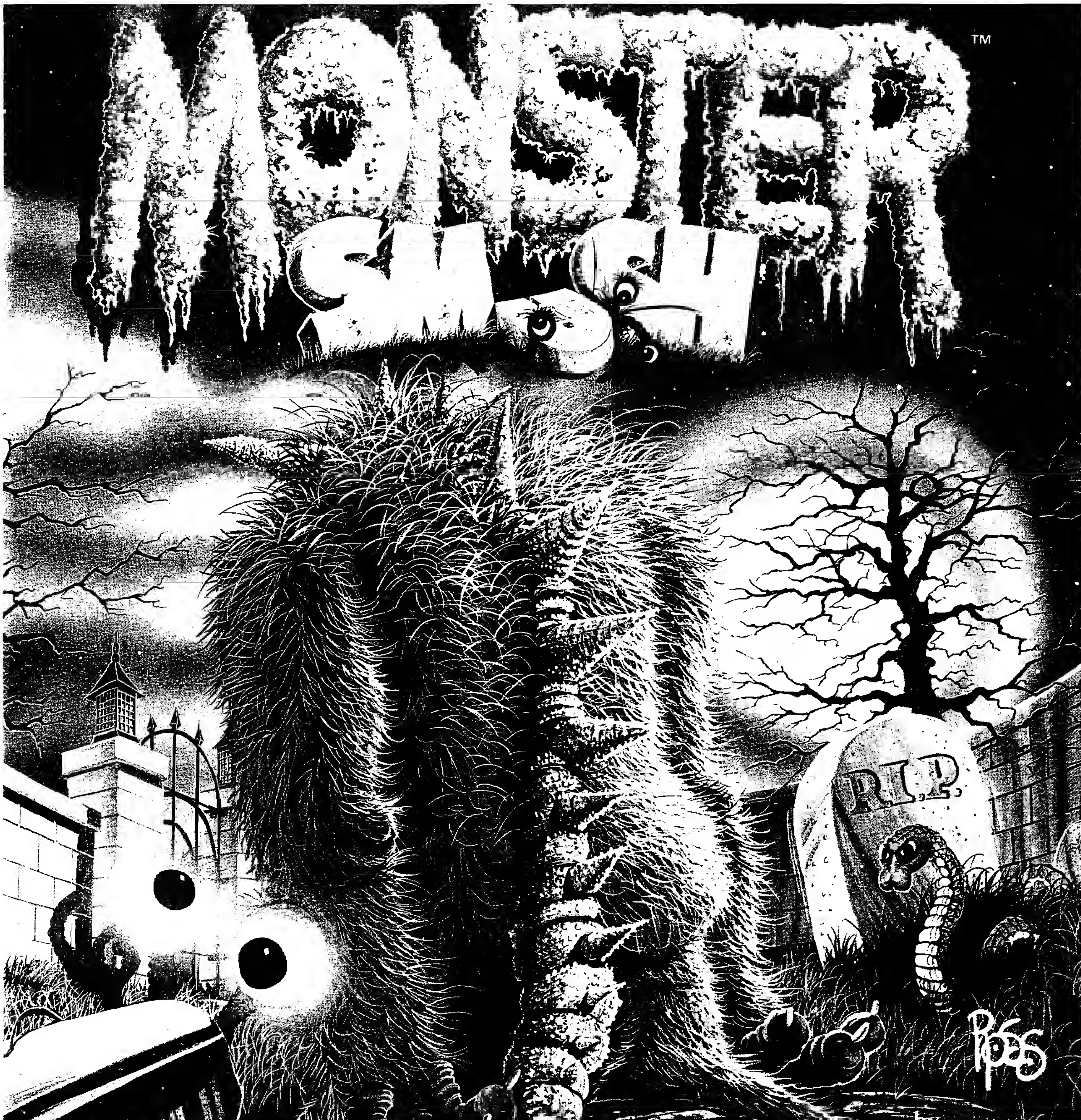
This is a poor man's interrupt addition to the Apple that requires only one wire! I have seen a few clock interrupt designs but none as simple and easy as this one.

The Apple has many periodic levels present on various points on the board. What if one of these was connected to the IRQ line of the 6502? There are four

(Continued on page 38)

The Apple II motherboard showing the locations of the pins where the jumper wire is to be continued.




MONSTER
SMASH
TM
R.I.P.
by
Dave
Eisler

counters (D11, D12, D13, and D14) that generate the horizontal byte position and vertical line position on the screen. Adding interrupts is easy. Connect Pin 11 of D11 to Pin 4 of H7/8 (the 6502). The best method for doing this is to use 30-gauge wire-wrap wire. Lift each chip slightly to allow the wire to fit in the socket along with the pin from the chip and reseat the chips and the wire. This wire then can be routed neatly along the board. By doing the modification this way you can remove it at any time.

The connection provides 59.92 interrupts per second on my Apple but may vary slightly on your own. (A Euro-Apple has different counter connections and will not work with this jumper.) This may seem high but the overhead in processing this many interrupt requests is only 2.9 percent.

There are several possible applications for a clock interrupt. The first is to keep track of time. The interrupt driver has an hours, minutes, and seconds counter that will maintain a running count after initialization of the rect hours, minutes, and seconds. Unfortunately, disks and other peripherals must inhibit interrupts so a few seconds are lost if disk usage is high. Without disk access this clock provides an exact method for determining keyboard response time. A second use is as a keyboard buffer to catch keystrokes during a running program; the interrupt driver also adds this function. Other uses are as print spoolers and foreground/background tasking.

Using the Interrupt Handler

Install the hardware modification as described and then enter the interrupt handler at $300. To enable interrupts from BASIC, CALL 768 (from monitor 300G). This toggles between interrupt enable and interrupt disable so the same CALL is used to disable interrupts. At the upper-right corner of the screen a space character will start blinking at the rate of once per second. This will tell you if interrupts are active or not. There are three counters that count hours, minutes, and seconds in 24-hour time. You can change this to 12-hour format by setting location $34C (844 decimal) to $0C (12 decimal). The other function of the interrupt handler is to buffer keyboard characters. From BASIC, enter a simple time consuming line such as

```
FOR I = 1 TO 5000: NEXT I
```

Enter a few characters while this is executing. When execution is finished, all characters entered will be input as though they were retyped. This can be used as a type-ahead buffer to allow programs to continue execution while still collecting input.

Program Description

With interrupts enabled, each interrupt causes execution to resume at the address of $FFFE upon completion of the current instruction. This is a pointer to the monitor interrupt service

routine at $FA40 ($FA86 in old monitors). The routine checks for a break condition and then does a JMP (indirect) to $3FE, which is the user-interrupt routine.

The first section of the interrupt handler installs the JMP vector in $3FE and then initializes the blinking character and buffer pointers. When an interrupt occurs, the registers must be saved. The accumulator is already saved in $45 by the break interrupt processing, so just the X and Y registers need to be saved. Since the IOSAVE routine in the monitor uses fixed locations to save the registers, it cannot be used in an interrupt routine. If a program that called IOSAVE was interrupted, then this second use of the save area would clobber the original saved registers. If IOSAVE used stack processing instead of fixed locations for a register save area, then it would be ideal. Since the stack needs to be inspected to determine the address of execution, I use fixed save locations.

The method used to derive seconds from a non-integral number of interrupts per second is called differential addition. A 3-byte counter (called COUNT) is used to maintain a running total. During each interrupt, a 3-byte value is added to the counter. This number is 16,777,216(2^{24})—divided by the number of interrupts per second. The carry is used as a once-per-second output. Since any extra remainder in the counter accumulates, this is an easy way to make integers count fractional values. A 24-bit counter represents an accuracy of 1.88 seconds per year. The drift of the master oscillator is then the limiting factor. The once-per-second output is added to the SECOND counter. Together with the MINUTE and HOUR counters, these count seconds, minutes, and hours. The three counters can be initialized to the time of day and will continue to keep time. When each second is counted, the upper-right screen location is set to a normal or inverse space, which is a discrete reminder that interrupts are enabled.

When a running program is not ready for data, the keyboard buffer subroutine accepts keystrokes and buffers them until required. To accomplish this, the return address of the running program is checked to determine whether or not it used the keyboard input routine. If it was in the keyboard input routine and there are previous characters in the buffer, then the next character is sent. To send a character, it is loaded into the accumulator, then the return address is modified to return to the end of the keyboard input routine, as though a character was entered from the keyboard. If the running program was not in the keyboard input routine, then any keystrokes are buffered. The buffer used is the normal keyboard input buffer ($200 - $2FF) but the interrupt handler uses the buffer from $2FF downwards while the keyboard input routine uses from $200 upward. This limits the number of characters that can be saved, but I have never had an overflow. Since the pulse used to create the interrupt is long, reentry of the interrupt routine would occur normally. A delay is added to the end of the routine to prevent this from happening and, although the interrupt routine is slowed down, usually it will not be noticed. The delay can be tested for the correct value by temporarily putting a LDA $C030 (AD 30 C0) at location $323, which will toggle the speaker without changing the timing each time the interrupt code is executed. Next change the delay to a lower value. If the sound changes as you type characters then the routine is being reentered.

There are three counters that count hours, minutes, and seconds in 24-hour time...The three counters can be initialized to the time of day and will continue to keep time.

Side Effects

One disturbing side effect occurs with programs that use the keyboard strobe ($C010) to check for a keypress. Since the interrupt handler clears the strobe for the next keystroke, the program usually will miss any keystrokes. One solution to this problem is to check BUFIN and BUFOUT. If they are unequal, a key has been pressed.

The second side effect involves the permanent connection of the clock interrupt when the interrupt handler is not loaded. On a machine with the old monitor ROM, everything will work fine until RESET is pressed. To return from the monitor, normally you type 3D0G. This will no longer work because 'G' resets the processor status register to the status saved during the break. The easiest solution to this problem is to type '48:4 N 3D0G RETURN', which sets the processor status interrupt disable 'on' so that 'G' will work. With the new monitor ROM, reset will operate normally. If you enter monitor you can use the same trick to exit.

Correcting the Clock

The Clock Calibrator program can be used to determine the correct entries for the interrupt timer counters. RUN the program with the Interrupt Handler installed and enabled, enter the correct time, and about 10 minutes later enter the correct time again. The program will calculate the correct settings for the counter and install them. For a more accurate clock, the interval between entries should be longer (overnight?); the day counter is included for this reason (day = 0 is defined as the same day). The clock shown on the display used for the Calibrator will stop updating when you begin an entry, but this will not affect accuracy. When the clock is accurate enough for you, don't forget to BSAVE the Interrupt Handler with the new counter values.

All you need now is a piece of wire!

Charles Putney received a BSME from Carnegie-Mellon University in 1973. Currently he is an engineer with Dataproducts Ltd, in Dublin and treasurer of the Dublin Apple Users Group. He may be contacted at 18 Quinns Road, Shankill, County Dublin, Ireland.

(Continued on next page)

About Apple Interrupt Listing

The listing for this article was generated by the new ORCA/M assembler from Hayden Software. The assembler includes a screen-oriented text editor and several excellent features such as macros, conditional assembly, upper/lower case, and a large subroutine library. The system is designed to have programs written in modules (subroutines) and has a link editor to link the modules together during assembly for a run-time package. Due to constraints imposed by our publishing programs for everyone to type into their own assembler, many of the more sophisticated procedures will not appear in print. (For additional information, see the June Operating System Data Sheet.)

There are a few points in the listing that might need clarification. Here is a list of differences among some of the standard assemblers. Those of you who use the monitor to enter programs will notice no difference.

KEEP	The name of the binary file generated
GEQU	A global equate necessary for zero-page variables
START	The beginning of a subroutine module
/	The high byte of a 16-bit number
DS x	Define storage space of x bytes, initialized to 0
DC I ' '	Define character of type 'I', enclosed in single quotes
	Types of letters (I) allowed:
	A Address
	B Boolean
	C Character constant
	D Double precision FP
	F FP number
	H Hexadecimal number
	I Two-byte integer
	J Four-byte Integer
	R One-byte integer

Apple Interrupt Listing

```
         KEEP INTHAND.OBJ
*        INTERRUPT HANDLER
*        WITH HOURS MINUTES SECONDS COUNTER
*        BLINKING CURSOR ONCE PER SECOND
*        KEYBOARD BUFFER
*
*        CHARLES H. PUTNEY
*
*        Copyright (C) 1983
*        by MICRO Ink
*        P.O.Box 6502
*        Amherst, MA 03031
*
*        PAGE ZERO ADDRESSES AND CONSTANTS
*
BUFINX GEQU  $34        KEYBOARD BUFFER INDEX
ACC    GEQU  $45        TEMPORARY STORAGE FOR ACCUMULATOR
*
STACK   EQU  $100       STACK PAGE
KEYBUF  EQU  $200       KEYBOARD INPUT BUFFER
INTVEC  EQU  $3FE       INTERRUPT VECTOR LOCATION
SCREEN  EQU  $427       UPPER RIGHT CORNER SCREEN LOCATION
KEYBD   EQU  $C000      KEYBOARD INPUT
KEYSTB  EQU  $C010      KEYBOARD INPUT STROBE
KEYIN   EQU  $FD1B      KEYBOARD INPUT
*
ENABLE GEQU  $20        INTERRUPT MASK FOR PROC STATUS REG
DELAY  GEQU  $39        DELAY TO MISS LONG INTERRUPT PULSES
INVRS  GEQU  $80        INVERSE CHARACTER MASK
SPACE  GEQU  $A0        SPACE
*
*------------------------------------
*
         START
         ORG   $300
*
*------------------------------------
*        INSTALL INTERRUPT VECTOR
*        AND ENABLE/DISABLE INTERRUPTS
*
```

(Continued)

Apple Interrupt *(continued)*

```
0300 A91A    INST  LDA  #INT       GET ROUTINE ADDRESS LOW
0302 8DFE03        STA  INTVEC     SET UP VECTOR
0305 A903          LDA  /INT       GET ROUTINE ADDRESS HIGH
0307 8DFF03        STA  INTVEC+1   SET VECTOR
030A ADBE03        LDA  BUFIN      MAKE BUFFER POINTERS EQUAL
030D 8DBF03        STA  BUFOUT
0310 AD1803        LDA  LO         GET ENABLE/DISABLE COMMAND
0313 4920          EOR  #ENABLE    MAKE INTO DISABLE/ENABLE COMMAND
0315 8D1803        STA  LO         PUT IT BACK
0318 78      LO    SEI             ENABLE/DISABLE COMMAND
0319 60            RTS
             *
             *------------------------------------
             *
             *     INTERRUPT HANDLER
             *
031A 8EC003  INT   STX  XREG       SAVE X (ACC ALREADY SAVED IN $45)
031D 8CC103        STY  YREG       SAVE Y
0320 18            CLC
0321 A202          LDX  #2         INITIALIZE INDEX
0323 BDC503  L1    LDA  COUNT,X    GET COUNT FOR THREE BYTE ADD
0326 7DC203        ADC  INTSEC,X   ADD TO INTSEC THREE BYTES
0329 9DC503        STA  COUNT,X    STORE RESULT BACK IN COUNT 3
                                   BYTES
032C CA            DEX             NEXT COUNT BYTE
032D 10F4          BPL  L1         NO
032F 9032          BCC  L3         NOT ONE SECOND YET?
0331 A93C          LDA  #60        GET 60 FOR COMPARE
0333 EEC803        INC  SECOND     BUMP SECONDS
0336 CDC803        CMP  SECOND     IS IS 60?
0339 D01D          BNE  L2         NO
033B A200          LDX  #0         GET A ZERO
033D 8EC803        STX  SECOND     INITIALIZE SECONDS
0340 EEC903        INC  MINUTE     BUMP MINUTES
0343 CDC903        CMP  MINUTE     IS IT 60?
0346 D010          BNE  L2         NO
0348 8EC903        STX  MINUTE     INITIALIZE MINUTES
034B A918          LDA  #24        GET 24 FOR COMPARE
034D EECA03        INC  HOUR       BUMP HOURS
0350 CDCA03        CMP  HOUR       IS IT 24?
0353 D003          BNE  L2         NO
0355 8ECA03        STX  HOUR       INITIALIZE HOURS
0358 ADBD03  L2    LDA  BLINK      GET BLINKING CHARACTER
035B 4980          EOR  #INVRS     INVERT IT
```

(Continued)

Apple Interrupt *(continued)*

```
035D 8DBD03        STA  BLINK     SAVE FOR LATER
0360 8D2704        STA  SCREEN    SHOW THAT INTERRUPTS ARE ON
0363 BA       L3   TSX            GET STACK POINTER
0364 E8            INX            SKIP OVER PROCESSOR STATUS
0365 E8            INX
0366 BC0001        LDY  STACK,X   GET PC LOW
0369 E8            INX
036A BD0001        LDA  STACK,X   GET PC HIGH
036D C9FD          CMP  /KEYIN    WAS IT IN KEYIN HIGH?
036F D02A          BNE  L5        NO
0371 C01B          CPY  #KEYIN    CHECK KEYIN LOW
0373 9026          8CC  L5        NO - TOO LOW
0375 C025          CPY  #KEYIN+10 CHECK KEYIN ROUTINE END
0377 B022          8CS  L5        NO - TOO HIGH
0379 ACBF03        LDY  BUFOUT    GET BUFFER OUT POINTER
037C CC8E03        CPY  BUFIN     ANY CHARACTERS IN BUFFER?
037F D00A          BNE  L4        YES - SEND ONE
0381 A9FF          LDA  #$FF      RESET BUFFER POINTERS
0383 8DBE03        STA  BUFIN
0386 8DBF03        STA  8UFOUT
0389 D024          BNE  L6        RELATIVE JUMP TO RESTORE REGS
038B B90002   L4   LDA  KEYBUF,Y  GET CHARACTER
038E 8545          STA  ACC       PUT IN ACC ON EXIT
0390 CEBF03        DEC  8UFOUT    GET READY FOR NEXT CHARACTER
0393 CA            DEX            NOW POINTS AT PC LOW
0394 A92E          LDA  #KEYIN+19 SET RETURN ADDRESS TO END OF
                                  KEYIN
0396 9D0001        STA  STACK,X   PUT IN STACK FOR PC LOW
0399 D014          8NE  L6        RELATIVE JUMP TO RESTORE REGS
039B 2C00C0   L5   8IT  KEYBD     KEY PRESSED?
039E 100F          BPL  L6        NO
03A0 AD00C0        LDA  KEYBD     GET KEYBOARD CHARACTER
03A3 8D10C0        STA  KEYSTB    CLEAR KEYBOARD STRO8E
03A6 AC8E03        LDY  BUFIN     GET BUFFER INPUT POINTER
03A9 990002        STA  KEYBUF,Y  SAVE THE KEY
03AC CEBE03        DEC  8UFIN     READY FOR NEXT TIME
03AF A239     L6   LDX  #DELAY    DELAY INSERTED TO AVOID RE-INTERRUPTS
03B1 CA       L7   DEX
03B2 DOFD          BNE  L7        NOT ZERO YET?
03B4 ACC103        LDY  YREG      RESTORE Y
03B7 AEC003        LDX  XREG      RESTORE X
03BA A545          LDA  ACC       RESTORE ACC
03BC 40            RTI            EXIT FROM INTERRUPT & RTS FROM
                                  KEYIN
              *
              *________________________________
              *
              *    INTERRUPT HANDLER VARIABLES
              *
03BD A0       BLINK  DC   I1<'SPACE' BLINKING CHARACTER
03BE 00       BUFIN  DS   1          BUFFER INPUT POINTER
038F 00       8UFOUT DS   1          BUFFER OUTPUT POINTER
03C0 00       XREG   DS   1          X REGISTER SAVE AREA
03C1 00       YREG   DS   1          Y REGISTER SAVE AREA
03C2 0445A9   INTSEC DC   H'0445A9'  (256)/(INTERRUPTS
                                     PER SECOND)
```

(Continued)

Apple Interrupt *(continued)*

```
03C5 000000   COUNT  DS   3          INTERRUPT TIMER
                                     (THREE 8YTES)
03C8 00       SECOND DS   1          SECONDS COUNTER
03C9 00       MINUTE DS   1          MINUTES COUNTER
03CA 00       HOUR   DS   1          HOURS COUNTER
03C8          ZZSIZE EQU  *-INST+1   PROGRAM LENGTH:
                     END
```

```
100 REM        Clock Calibrator
120 REM
130 REM        Charles H. Putney
140 REM         18 Quinns Road
150 REM         Shankill
160 REM         Co. Dublin Ireland
170 REM
175 REM       Copyright (C) 1983
180 REM       by MICRO Ink
185 REM       P.O.8ox 65Ø2 Amherst, NH Ø3Ø31
19Ø TEXT: HOME
2ØØ DØ = Ø
21Ø VTAB 5: HTAB 6: INPUT "ENTER TIME
     HH,MM,SS ";HØ,MØ,SØ
22Ø POKE 97Ø,HØ: POKE 969,MØ: POKE 968,SØ:
     POKE 965,Ø: POKE 966,Ø: POKE 967,Ø
23Ø HOME
24Ø I = PEEK (964) + 256 * PEEK (963) + 65536 * PEEK (962)
25Ø F = 256 ↑ 3 / I
26Ø VTAB 5: HTAB 1Ø: PRINT "FREQUENCY = ";F
27Ø HC =  PEEK (97Ø)
28Ø MC =  PEEK (969)
29Ø SC =  PEEK (968)
3ØØ IF HC = 23 THEN Q = 1
31Ø IF Q = 1 AND HC = Ø THEN DC = DC + 1:Q = Ø
32Ø VTAB 15: HTAB 13
33Ø IF HC < 1Ø THEN PRINT "Ø";
34Ø PRINT HC;":";
35Ø IF MC < 1Ø THEN PRINT "Ø";
36Ø PRINT MC;":";
37Ø IF SC < 1Ø THEN PRINT "Ø";
38Ø PRINT SC;
39Ø VTAB 2Ø: HTAB 6: PRINT
     "ENTER CORRECT TIME DD,HH,MM,SS ";
4ØØ IF PEEK (958) = PEEK (959) THEN 27Ø
41Ø INPUT D,H,M,S
42Ø HC = PEEK (97Ø):MC = PEEK (969):SC = PEEK (968)
43Ø EC = DC * 864ØØ + HC * 36ØØ + MC * 6Ø + S - DØ
     * 864ØØ - HØ * 36ØØ - MØ * 6Ø - SØ
44Ø ER = D * 864ØØ + H * 36ØØ + M * 6Ø + S - DØ *
     864ØØ - HØ * 36ØØ - MØ * 6Ø - SØ
45Ø I = I * ER / EC
46Ø POKE 962,I / 65536
47Ø IL = ((I / 65536) - INT (I / 65536)) * 65536
48Ø POKE 963,IL / 256
49Ø POKE 964,IL - 256 * INT (IL / 256)
5ØØ POKE 965,Ø: POKE 966,Ø: POKE 967,Ø
51Ø HØ = H:MØ = M:SØ = S
52Ø GOTO 22Ø
```

MICRO

INTERFACING THE ATARI JOYSTICK

How to use your Color Computer joystick inputs and hook up an Atari switch joystick to the Color Computer.

by John Steiner

While playing *Ghost Gobbler* on the Color Computer, I was frustrated by the lack of positive action on the joystick. An analog joystick usually is sufficient for operation, but there are times when a positive-switch closure is better. This article shows you how to interface the popular Atari joystick to the TRS-80C. Also included is an applications program that demonstrates the differences between the two joysticks.

The Color Computer has an interesting joystick interface, and the analog joystick port is an excellent feature of CC. With a minimal investment, you can have the best of both worlds by following the modification procedure and changing a few programming techniques. Many programs written in BASIC can be modified easily to use the switch joystick.

How the Joystick Works

First take a look at the Radio Shack joystick interface (see schematic in figure 1). There are two potentiometers; they are labeled X for horizontal movement, and Y for vertical movement. The high sides of the pots are connected to +5 volts, and the low

Atari Joystick requires:
TRS-80C
Atari Joysticks
some hardware

Figure 2: **Block diagram joystick circuitry.**

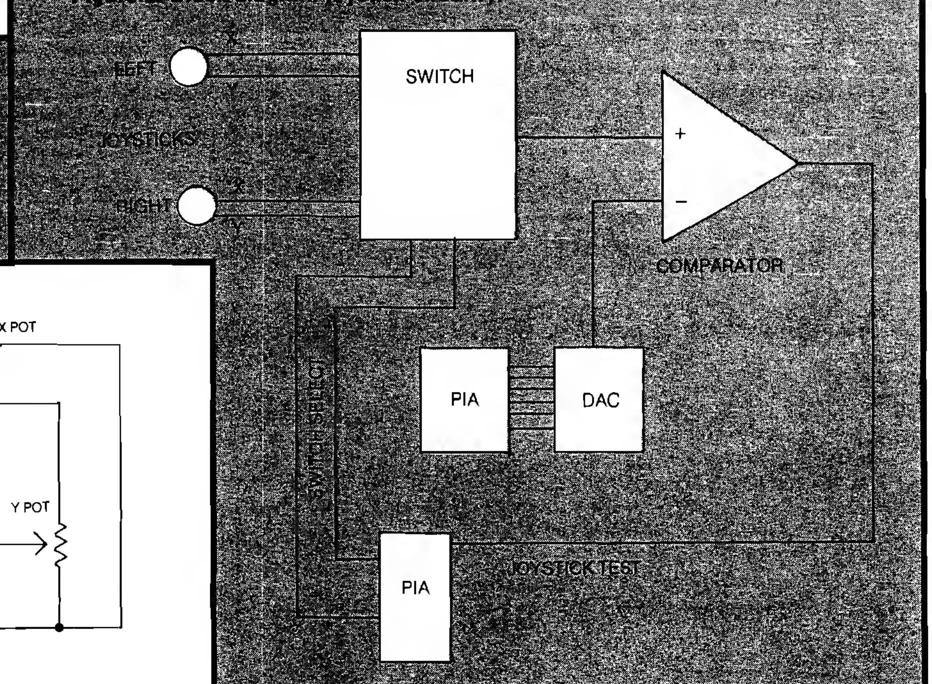
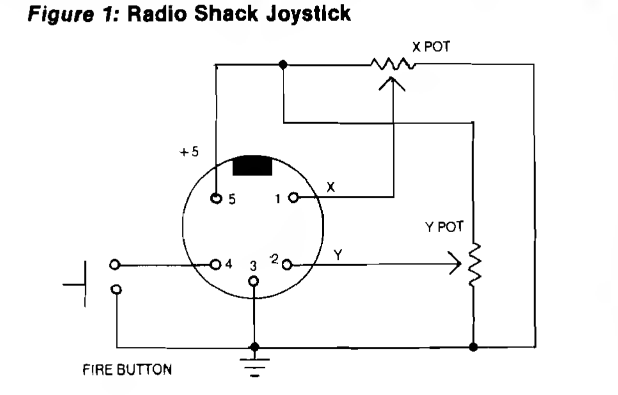


Figure 1: **Radio Shack Joystick**



sides are connected to ground. The control wipers are connected to the joystick inputs inside the computer and the fire button is a normally open switch that connects to ground when pressed.

Look at the joystick schematic. You can change the voltage at pin 1 from approximately zero volts to +5 volts by moving the joystick from left to right. Similarly, pin 2 will vary between zero volts and +5 volts if you move the joystick from up to down.

Figure 2 is a block diagram of the internal joystick interface. The joysticks are connected to an electronic switch. When the joysticks are read, the correct input is selected from the switch and sent to a comparator. (More on this later.) To read the joystick, a machine-language subroutine is called. Here is how it works: the switch is connected to either pin 1 or 2, left or right, depending on the reading specified in the program. A peripheral interface adapter (PIA) inside the computer sends data to a digital-to-analog convertor (DAC). To understand more clearly, you could start by storing a zero in the PIA. A zero into the DAC outputs approximately zero volts to the comparator. The function of the comparator is to put out a zero as long as the voltage at the minus input is lower than the voltage at the plus input. Increase the PIA number by one, which increases the output voltage of the DAC, then check to see if the output of the comparator is a logic one. If not, increase the PIA by one again. There are 64 discrete voltages available from the DAC; when the comparator output is finally one, the voltage will be within 5/64ths of the joystick output voltage.

The above method works, but it is not linear in speed. For example, lower voltage outputs could be found much faster than higher voltages. Indeed, if the joystick is all the way down, or all the way right, 64 readings would have to be made. Radio Shack decided to use an iterative, or binary, search. The joystick routine needs only six samples to identify any of the 64 possible joystick values. The routine does this by starting at 32 and either dividing the number in half or dividing in half and adding to the original number, depending on the output of the comparator. Check a programming manual for more on iterative search routines. Once the value is found (always between 0 and 63, inclusive), it is stored in a variable by BASIC.

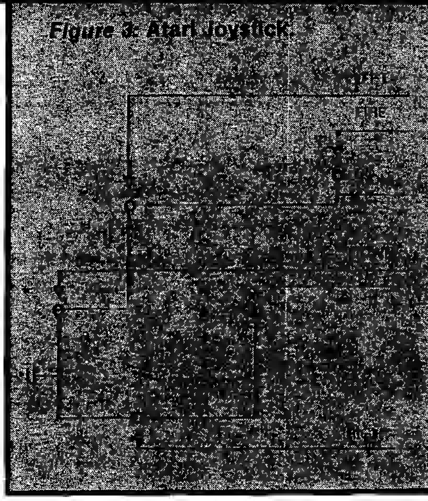


Figure 3: Atari Joystick

How to Use the Joystick in a BASIC Program

Color BASIC has a special JOYSTK command that reads the output of the desired joystick. Its syntax is A = JOYSTK(X). The variable A can be any numeric variable normally allowed. When the command is complete, A will contain the value read at the joystick port. The variable X can be 0, 1, 2, or 3 only. X defines the joystick parameter being accessed: JOYSTK(0) is the right X or horizontal reading, (1) is the Y or vertical reading, (2) is the left joystick X, and (3) is the left joystick Y. There is one constraint when using the joystick command: you must always read JOYSTK(0) before reading any other joystick value, whether or not you plan to use the value. For example, if you wish to read the vertical value of the left joystick only, do the following:

```
10 A = JOYSTK(0)
20 A = JOYSTK(3)
```

variable A will finally contain the value of JOYSTK(3) and can be used as required.

Since the joystick is an analog input, you can use almost any device as a joystick input. A thermistor could turn the computer into a thermometer, or a photo detector could become a simple light pen. See William Barden's "Color Computer from A to D," *BYTE*, December 1981, for further applications using the joystick ports and a description of the circuitry and JOYSTK routines. Spectrum Projects offers a light pen that uses the joystick input.

The Need for a Switch Joystick

Although the analog joystick port is an excellent feature, sometimes the large range of numbers available and the amount of play in the stick itself make directing the stick difficult. The listing included is a case in point (the program is an exercise in using the port in a BASIC program). Enter the program and run it. If you use the standard joystick, you will see what I mean — it is difficult to start the gunner at the bottom and when he begins to move he is hard to stop, or he moves in the other direction.

The joystick calculations are handled in lines 280 to 320, variable PP is a PRINT @ position, and the joystick values modify the position, one position at a time. The program needs to know only whether the joystick is left of center or right of center. Lines 290 and 300 check the position. The sensitivity to the joystick can be varied by changing numbers 52 and 12.

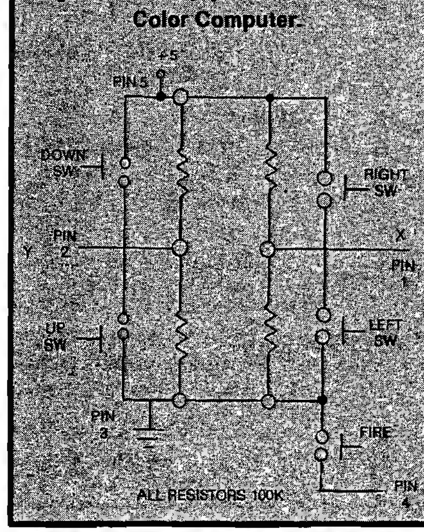
There are other methods of using the joystick to indicate position. One way to set the position of the gunner is to read the joystick and divide its value by 2. This will give you a range of 0 to 31. The position of the gunner can then be set directly, as there are 32 PRINT @ positions on the gunner line. This technique is difficult to use, however. Slight movements of the joystick result in large movements of the gunner.

Hooking Up and Using the Atari Joystick

The alternative to a linear joystick is a digital or switch joystick as found

(Text continued on page 47)
(More figures and listings page 46)

Figure 4: Atari joystick wired for Color Computer.

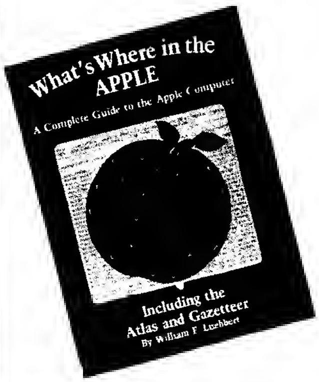


PRODUCTS from MICRO

Revised edition now available! All new text added to the original atlas and gazetter...

"What's Where in the Apple"

A Complete Guide to the Apple Computer



This revised edition of the famous Apple Atlas will:

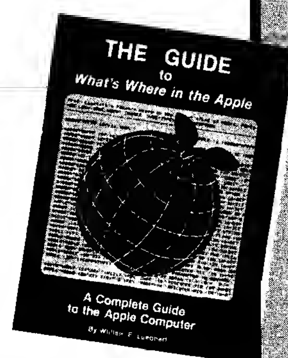
- provide you with more information on the Apple's memory than is available elsewhere
- guide you with a numerical Atlas and an alphabetical Gazette — to over 2000 memory locations of PEEKs, POKEs and CALLs
- give you names and locations of various Monitor, DOS, Integer BASIC and Applesoft routines and tells you what they're used for
- enable you to move easily between BASIC and machine language
- guide you through the inner workings and hidden mechanisms of the Apple

The Atlas and the All New Guide are now available in one 256-page Wire-O-Bound book for only $24.95.

All Apple users will find this book helpful in understanding their machine and essential for mastering it.

Retail price $24.95

If you own the original **What's Where in the Apple** purchase **The Guide** to complete your edition. This 128-page Wire-O-Bound version contains all the new material to be used with the memory map and atlas.



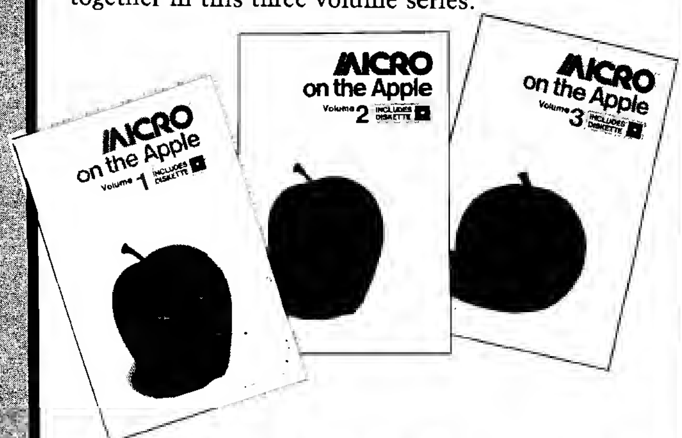
Don't miss this opportunity to complete your edition of **What's Where in the Apple.**

Order your copy of The Guide now.

Retail price $9.95

Get the most out of your Apple with the "MICRO on the APPLE" series...

This series of books for the APPLE computer is the best source of Apple specific programming aids, utilities, enhancements, games and much more. MICRO has taken the best APPLE material ever published in MICRO magazine and brought it together in this three volume series.



Volume 1 allows you to —

Round and format numbers accurately, get lower-case letters and punctuation into Applesoft strings or play the hit game "Spelunker"

Volume 2 lets you —

Speed up machine-language programming using five powerful machine-language aids, add additional editing and I/O features, or play the intriguing game "GalactiCube"

Volume 3 gives you more —

Machine-language aids, I/O enhancements, graphics and games

Each volume comes complete with diskette to save you the time of typing hundreds of lines of code.

If you use an APPLE computer you should own these books.

Retail price $24.95

FOR EASY ORDERING
CALL TOLL FREE
1-800-345-8112
(IN PA 1-800-662-2444)
MasterCard and VISA Accepted

AND now the "MICRO on the APPLE" series is available as a boxed set.

All three volumes are packaged in an attractive box with the 110 programs on one diskette (double-sided).

The set is available at a **20% savings** off the individual price.

Retail price $59.95

Figure 5

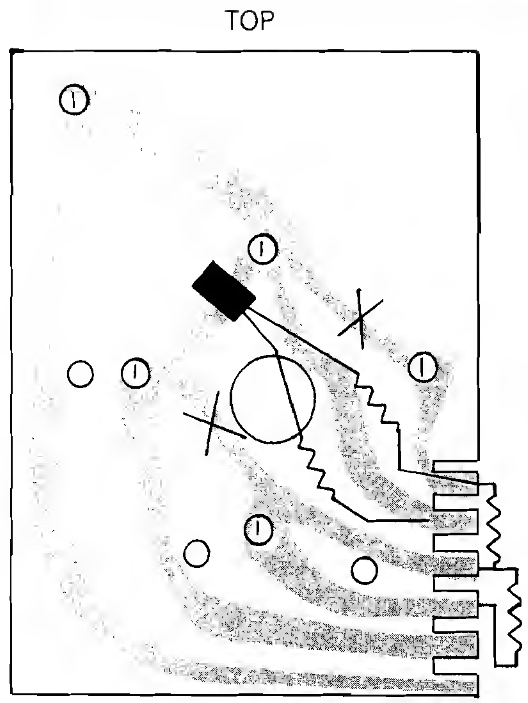
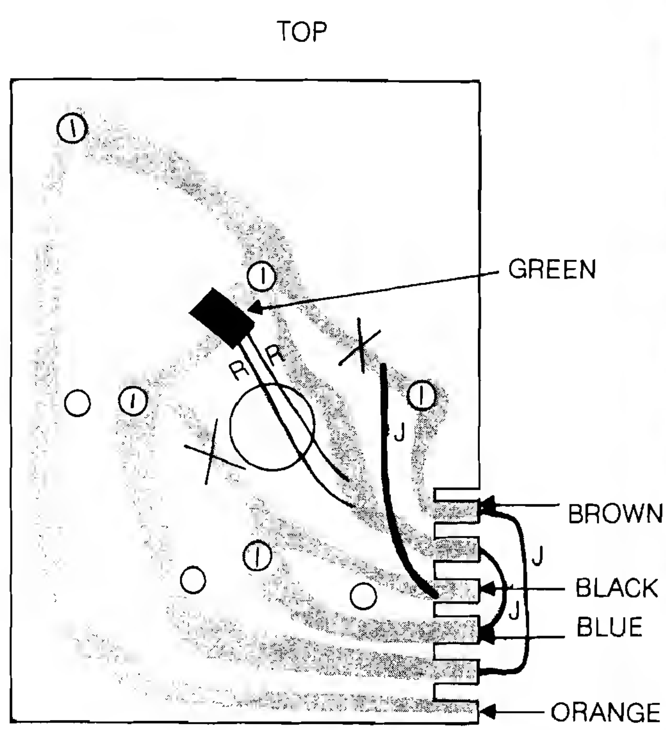


Figure 6



J = JUMPER WIRE   R = RESISTOR

Listing 1

```
1 'TARGET
2 'JUNE 15, 1982
3 'BY JOHN STEINER
4 'WRITTEN TO DEMONSTRATE THE ATARI
   JOYSTICK ON THE TRS-8Ø C
1Ø CLEAR 5ØØ
2Ø GOSUB 85Ø
3Ø M1$ = "O2;L2;GO3;D;L8;C;O2;B;A;O3;
   L2;G;L4;D;L8;C;O2;B;A;O3;L2;G;L4;D;L8;C;O2;
   B;O3;C;L2;O2;A"
4Ø PLAY "T4;XM1$;"
5Ø M2$ = LEFT$(M1$,15)
6Ø M3$ = "O2;L4;G;G;O3;E;L1;E;L4;E;E;F;L8;
   F;E;L1;D;L4;D;O2;G;G;O3;E;L1;E;L4;E;G"
7Ø TI=Ø : TIMER=Ø : PC=Ø : X=1 : CO=134 : CL=137
8Ø FOR I=1 TO 32 : X$ = X$ + CHR$(128) : NEXT
9Ø PC$(X)=""
1ØØ PRINT @ 224,"HOW GOOD ARE YOU?";
11Ø PRINT" ENTER 1 IF YOU ARE GOOD, 4 IF YOU'RE NOT";
12Ø A$=INKEY$ : IF A$ = "" THEN 12Ø
13Ø DI=VAL(A$) : IF DI< OR DI>4
    THEN PRINT" ENTER 1 TO 4 ONLY"; : GOTO 1ØØ
14Ø DI=DI*5Ø
15Ø CLS(Ø)
16Ø FOR J=1 TO 63
17Ø SET(J,3Ø,1)
18Ø NEXT J
19Ø '
2ØØ P$=CHR$(128) + CHR$(151) + CHR$(155) + CHR$(128)
21Ø E$=CHR$(128) + CHR$(CO) + CHR$(CL) + CHR$(128)
22Ø D$=CHR$(128) + CHR$(128) + CHR$(128) + CHR$(128)
23Ø FOR I=1 TO 7 : PC$(X)=PC$(X) + E$ : NEXT
24Ø PRINT @ 448,X$;
25Ø PRINT @ PP,P$;
26Ø FS=PEEK(6528Ø) : IF FS=126 OR FS=254 THEN 37Ø
27Ø TI=TIMER : PRINT @ 48Ø,"TOTAL SCORE =";SC;
28Ø ZP=JOYSTK(Ø)
29Ø IF ZP>52 THEN ZZ=ZZ+1
3ØØ IF ZP<12 THEN ZZ=ZZ-1
31Ø IF TI>DI THEN GOSUB6ØØ
32Ø PP=ZZ+448:IFPP>476THENPP=476
33Ø IF PP<448 THEN PP=448
34Ø PRINT @ PC,PC$(X);
35Ø IF PC>416 THEN 53Ø
36Ø FS=PEEK(6528Ø) : IF FS=126 OR FS=254
                     THEN 37Ø ELSE 24Ø
37Ø J=27
38Ø PLAY"T255;XM2$;"
39Ø FOR I = Ø TO 63 : F=POINT(I,28)
4ØØ IF F=2 THEN 43Ø
41Ø NEXT
42Ø IF J=Ø THEN 24Ø
43Ø SET(I,J,8)
44Ø J=J-1
45Ø RESET(I,J+1)
46Ø IF POINT(I,J)<>Ø THEN 48Ø
47Ø GOTO 42Ø
48Ø GOSUB 77Ø
49Ø PLAY"T255;XM3$;"
5ØØ SC=SC+1
51Ø IF PC$(X)=D$ + D$ + D$ + D$
    + D$ + D$ + D$ THEN GOSUB 64Ø
52Ø GOTO 24Ø
53Ø FOR I=1 TO 32 : PRINT @ PC+I,CHR$(128); : NEXT
54Ø FOR I=1 TO 32 : PRINT @ 447+I,CHR$(128); : NEXT
55Ø SC=SC-5
56Ø IF X>5 THEN 64Ø
57Ø CO=CO + 16 : CL=CL + 16 : TI=Ø :
    TIMER=Ø : PC=Ø:X=X + 1 : PC$(X)=""
58Ø GOTO 19Ø
59Ø ' PRINT NEW ENEMY POSITION
6ØØ FOR I=1 TO 28 STEP 4
61Ø PRINT @ PC+I,D$;
62Ø NEXT
63Ø PC=PC+32 : TIMER=Ø : RETURN
64Ø IF X<5 THEN SC=SC+1Ø : GOTO 53Ø
65Ø PRINT @ 26Ø,"YOUR TOTAL SCORE IS";SC;
66Ø IF DI=2ØØ THEN BP=SC*1Ø
67Ø IF DI=15Ø THEN BP=SC*25
68Ø IF DI=1ØØ THEN BP=SC*5Ø
69Ø IF DI=5Ø THEN BP=SC*1ØØ
7ØØ PRINT @ 292,"BONUS POINT TOTAL IS";BP;
```

Listing 1 *(continued)*

```
710 FOR [illegible]
720 PRINT [illegible]
730 [illegible]
740 [illegible]
750 IF [illegible]
760 [illegible]
770 IF [illegible]
780 IF [illegible]
790 IF [illegible]
800 IF [illegible]
810 [illegible]
820 IF [illegible]
830 IF [illegible]
840 RETURN
850 CLS : PRINT [illegible]
    (Y/N) [illegible]
860 I$=INKEY$ [illegible]
870 IF I$="N" [illegible]
880 CLS : PRINT [illegible]
890 PRINT [illegible]
900 PRINT"DESTROY THE [illegible]
910 PRINT [illegible]
    PRINT"POSITION [illegible]
920 PRINT [illegible]
930 PRINT [illegible]
940 PRINT [illegible]
    PRINT [illegible]
950 PRINT @ 448, [illegible]
960 I$=INKEY$ [illegible]
970 CLS(0) : RETURN
```

(Continued from page 43)

on the Atari home computer and video games. The joystick has five switches, each of which close independently as the joystick is used. Figure 3 is a schematic of the Atari joystick circuitry as it is wired originally. For our application, the actual pin numbers are not required and therefore are not listed. All the switches are connected to ground, unlike the TRS-80C joystick, which needs both a +5 volt and ground hook-up.

Normally all inputs are high until a switch is closed. When a switch closes, a corresponding pin on the cable jack goes low and the software notes the changed state of the port. The switch joystick is not as versatile, but is useful when you are looking for digital changes. In my program, I need to know only whether the operator wants to go left, right, or stay where he is.

If you wire the joystick as shown in figure 4, the joystick port receives only three values: the center position (no switch closed) returns a value of 32; a left command returns a zero; and a right command returns a 63. You can do this by rewiring the joystick and adding a voltage divider network to the X and Y switches accordingly. The modification requires only four 100K resistors, a 5-pin DIN connector to fit the computer, and a joystick. Converting one joystick costs about as much as one new Radio Shack joystick.

Disassembling the stick is easy; just remove the four phillips screws from the bottom. Be careful when pulling the stick apart — there is a tiny spring on the fire button that could get lost easily. Figure 5 is a drawing of the printed circuit board inside the joystick. The switches are mounted directly on the circuit board and are only metal contacts soldered to the foils, which are identified in the diagram by lines drawn through circles. The fire button is in the upper left corner, and north, east, west, and south are where they should be. A row of slots on the lower right of the board is used to connect the cables.

The figure also has two Xs marked over foil traces. Using a razor blade, cut the foil at those points. You can double check it with an ohmmeter, if you have one. You will also have to slice through the plastic membrane covering the circuit board. Cut about ¼" of plastic away from either side of the break. This allows you to solder a couple of wires to the foil. (Figure 5 shows the location of the four resistors.)

When you finish cutting the foil you are ready to begin wiring. Figure 6 contains a diagram of the jumper wires that must be soldered to the board. The process is not difficult; I was able to complete the wiring in about an hour, which also included figuring out how to cut the board. I mounted all the resistors on the bottom side of the board and ran leads through the slots and large center hole. There are two jumper wires and four resistors to be installed (see figure 6).

When you finish installation, unplug the white wire from the circuit board and cut it off close to where the cable comes into the stick; it is no longer needed. Then unplug the green wire and cut off the connector. Solder the wire to the foil at the same point as the two resistor leads, as shown in figure 6. All the other colors are left as originally found. Those colors are listed in the figure in case your joystick has a different color combination.

Now you are ready to install the DIN plug. Cut off the Atari plug close to the plug end and strip the outer cable and inner conductors. Cut off the white cable at the cable and connect the wires as follows:

Table 1

Pin	Color
1	brown
2	blue
3	green
4	orange
5	black

Table 1 shows the pin numbering from the solder side of the connector. Connect the colors to the pin numbers as shown. When you are done, enter the program shown in listing 2 to check the operation of the new joystick.

Listing 2

```
10 CLS
20 A = JOYSTK(0)
30 B = JOYSTK(1)
40 PRINT @ 224, "HORIZ = ";
   A,"VERT = ";B;
50 GOTO 20
```

Enter the program listing and install the joystick in the right joystick port. When you run listing 2, the two numbers should be 32. Moving the joystick left changes HORIZ to 0, moving it right returns a 63. Moving the stick up changes VERT to 0, and moving it down changes it to 63. Changing the program to JOYSTK(2) and (3) in lines 30 and 40 tests the left joystick port.

Now get out a few of your joystick programs: some will show a noticeable improvement, other will not. Don't worry about causing damage if you miswire the connector — the port is protected with a current-limiting resistor. Even connecting the +5-volt pin to ground is harmless, as there is only about 50 mA of current flowing.

This modification is easy and well worth the effort. If you have trouble, send a self-addressed, stamped envelope and I will answer any questions. The program listing is available by sending a blank cassette and enough postage for shipping. Now that you know how, get off the "pot" and get on the "stick."

John Steiner is an electronics instructor in the Fargo Public School System. He has a 32K Color Computer with disk, and an Epson MX-80 printer. You may contact Mr. Steiner at 508 Fourth Ave., N.W., Riverside, ND 58078.

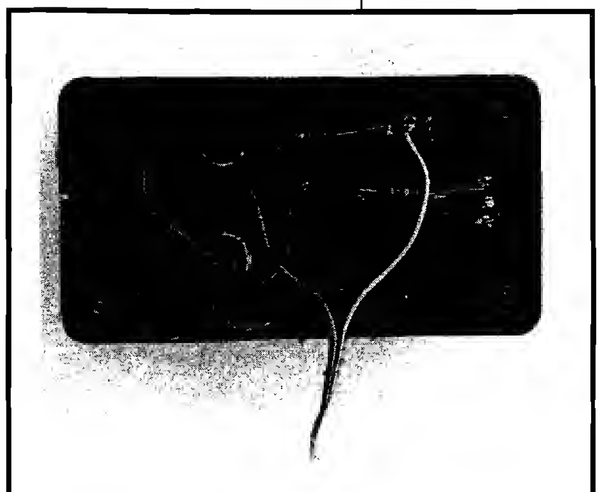
MICRO

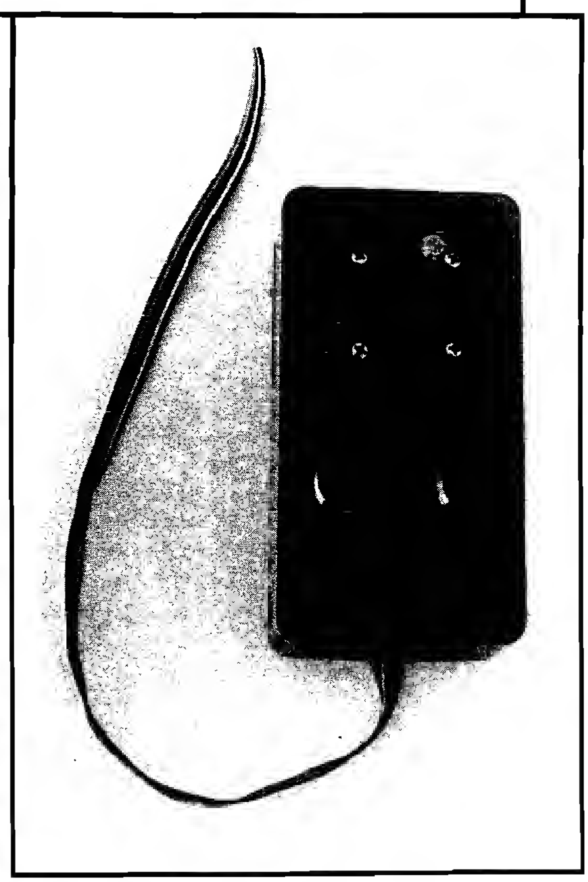
An Inexpensive Joystick for the Apple II

by **Phil Daley**
Circuit by Dan Weston

Apple Joystick
requires:
Apple II
and hardware parts

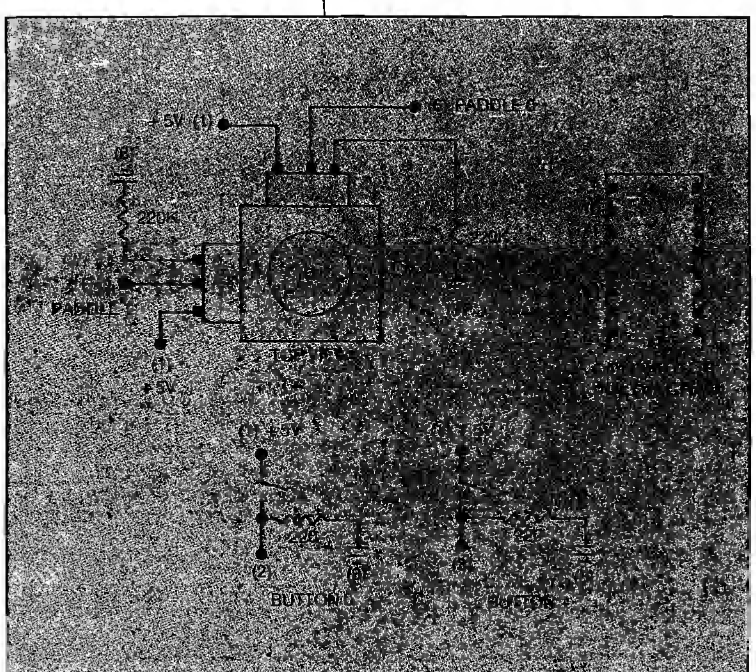
Construct your own joystick for the Apple II at a savings of over 50%. A wiring diagram is included.





The only knowledge necessary for you to construct your own joystick is the use of a few simple tools and a soldering iron. In addition to the parts list below, you will need a piece of ribbon cable (six strands) or six pieces of wire. Separate wires are better if you plan to use the stick a lot; ribbon cable is likely to break internally under the strain of much flexing.

Top right: Interior of assembled joystick. *Bottom right:* Completed joystick. *Below:* Circuit Diagram.



The first step is to visit your local Radio Shack outlet and buy the following parts:

1	100K Joystick pot	271-1705	$4.95
2	220K ohm resisters	271-049	19/pr
2	220 ohm resistors	271-015	[illegible]/pr
2	Push buttons	275-609	$1.39/pr
1	Project case	270-222	$2.19

Also you will need a 16-pin DIP header to plug into the game paddle socket. Unfortunately RS no longer stocks these parts, but most electronic parts stores should carry them.

The first step is to drill two 3/8" holes for the buttons. Be sure to place them where they will be convenient for playing games that require both buttons. Then drill, or cut, a larger 1" hole for the joystick. This hole will require four smaller holes, spaced evenly around the edge, for the mounting screws. The safest way to make sure they are in the proper location is to make a paper pattern to place on the box to mark the drilling spots.

Next, fasten the joystick to the box with the four screws and each button to the box with the washer and nut supplied. Solder a wire from each of the connections marked '1' to the next connection marked '1'. The order is not important as long as they are all connected to each other. Solder one end of each of the 220K ohm resistors to the appropriate joystick connection (#8) and one end of each of the 220 ohm resistors to the #8 connection on each button. Solder all the free ends of the four resistors together in the center of the case.

Cut a small channel in one end of the case for the ribbon cable or the separate wires to fit through when the case cover is in place. We used a razor-blade knife to remove enough of the plastic so that the cable wouldn't get crushed when the cover was tightened down. Solder four of the six wires to the proper location, #'s 2,3,6, and 10. The two remaining wires connect to locations previously soldered. One connects to the resistors connection in the center and the other connects to any one of the locations numbered '1'.

Solder the other end of the wires to the proper location on the DIP header that has the numbers printed on it. The number 1 location is on the bottom right looking down from above. Run the cable through the channel, put the four cover screws back into the case, and you are all set to plug the header into the game paddle socket.

We think that you will find games allowing use of a joystick instead of the keyboard A/Z and forward/backward arrow maneuvers will be a lot easier to control with this inexpensive joystick. Happy gaming!

Many thanks to John Morse of Newton, MA, for help with the resistor values for the circuit.

Dan Weston teaches a self-contained eighth grade in Brooks, Oregon. He may be contacted at 195 23rd NE, Salem, OR 97301. Phil Daley is a Apple editor for MICRO magazine.

MICRO

PET GOES ROM

by Jerry D. Brinson

Software and hardware are provided for a PET 2532 EPROM Programmer.

You have probably noticed some extensions to the PET operating system for sale. WORDPRO, TOOLKIT, etc. are systems that use a portion of the 12K ROM memory block not used by the PET operating system. It would be nice to use some of that free area for your own machine-language programs or to store your favorite BASIC program in ROM. This article shows you how.

First you must choose a good EPROM. Texas Instruments makes an EPROM that's pin-for-pin compatible with the three empty ROM sockets that lie underneath the hood of a 16/32K PET. That EPROM is the TI-2532 and is matrixed 4K by 8 bits. You will need one TI-2532 for each 4K of ROM that you use.

Next you must program the EPROM. The circuit in figure 1 and the program in listing 1 does the trick. It may not be fast but it's cheap and it works!

All of the parts can be purchased at an electronics store. I used a 44/22-pin experimenter's board to lay out the circuit mentioned. I used sockets for all the integrated circuits, although only the EPROM socket is necessary. This should be a ZIF socket or some other type that allows quick and easy removal of the EPROM.

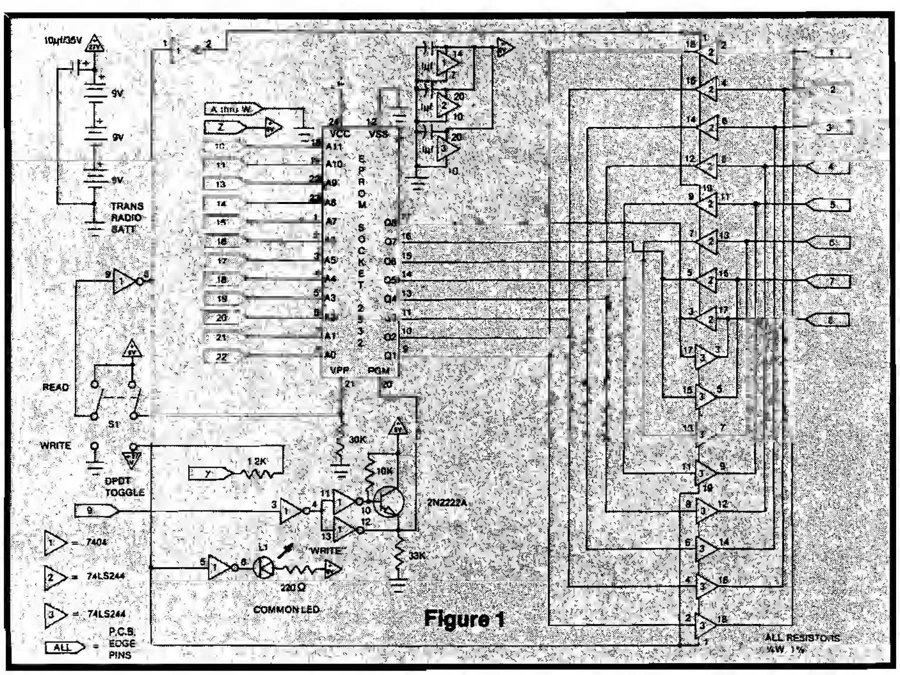


Figure 1

Once the circuit board is finished, a harness to interface the 2532 programmer to the PET must be made (see figure 2). Note that both the user port and the IEEE-488 port are used. Cable length should not exceed 18 inches. When the circuit board and the interfacing harness have been built, a check of all connections with an ohmmeter is suggested.

Listing 1 must now be entered into the PET. Double check that what is in your PET's memory equals listing 1. You should save this program on tape/disk for future use.

The program provides a menu to allow various functions to be called. Other than the PROGRAM function, the READ and COMPARE functions are used to check what you actually placed into the EPROM. The EXIT TO MONITOR and END functions are self explanatory.

The 2532 Programmer has two areas of RAM that are reserved. The RAM area from 8192 ($2000) to 12287 ($2FFF) is reserved for the program that you wish to enter into EPROM. The RAM area 12288 ($3000) to 14335 ($3FFF) is reserved to store what is read from the EPROM.

Prior to running the 2532 Programmer in the PROGRAM mode you must, of course, have what you want to place into the EPROM in the 2532 programming area. For machine-language programs this is rather easy. By using the PET monitor you are able to LOAD and SAVE programs to any spot in memory. For BASIC the process is somewhat different. First I will discuss machine-language programs.

The ROM area where the EPROM program will reside is located at 36864 ($9000) to 49151 ($BFFF). This is broken into three 4K blocks.

Block 1: 36864 ($9000) to 40959 ($9FFF)
Block 2: 40960 ($A000) to 45055 ($AFFF)
Block 3: 45056 ($B000) to 49151 ($BFFF)

The empty ROM sockets in the PET, located from left to right, are Block 3, Block 2, and Block 1. You can store up to a 12K continuous program in three 4K blocks.

The first requirement is to select the ROM area that you want to use. If

you own WORDPRO or TOOLKIT part of the decision has already been made for you, as these programs reside in Block 3. Assume, for example, that you have a 2K machine-language program and you want this program to reside in Block 1, from 36864 ($9000) to 38911 ($97FF). To work with the 2532 Programmer the machine code should be loaded into RAM at location 8192 ($2000) to 10239 ($27FF). Note that although the program is loaded into the RAM location, the machine-code pointers, jumps, etc. should be as if the program were located in ROM at the target address you have chosen. Any programs less than 4K will afford extra storage for other programs. You may stack programs; otherwise you should fill the remaining RAM programming area with ($FF) decimal 255. This will allow you to add future programs to the EPROM without erasing.

Insert a TI-2532 into the proper socket on the programming board. *Note:* Three 9-volt transistor radio batteries are used to supply the necessary 25-volt supply. *Always* disconnect the 25-volt supply at the same time you disconnect the 5-volt supply. Or disconnect the 25-volt supply prior to disconnecting the 5-volt supply; otherwise damage may result to the EPROM.

Load the 2532 Programmer into the PET and RUN. The PET will tell you what steps to take. Select the READ function to check what was put in EPROM. When the READ function is finished, select the COMPARE function. This will make a comparison between what you intended to put into ROM and what actually got there. If everything goes well the COMPARE function will finish without an error message.

Turn off the PET and insert the EPROM into the proper socket, noting correct polarity. Turn the PET back on and type SYS xxxx with xxxx equal in decimal to the entrance of your machine-language program. Your program should execute properly. Note that at the end of your routine a RTS should be included to return you to BASIC.

Most BASIC programs will run in ROM unless they are self-modifying. The program in listing 2 will take a BASIC program and store a modified copy in the 2532 programming area. You should enter the program in listing 2, check and SAVE it, then RUN the program. It will make a copy of itself in the 2532 programming area ready to be placed in ROM. This is necessary so that you may convert other programs using the Relocator from ROM. If you have greater than 16K memory, you may store the Relocator in upper RAM instead by calling a RAM address instead of a ROM address when asked during program execution.

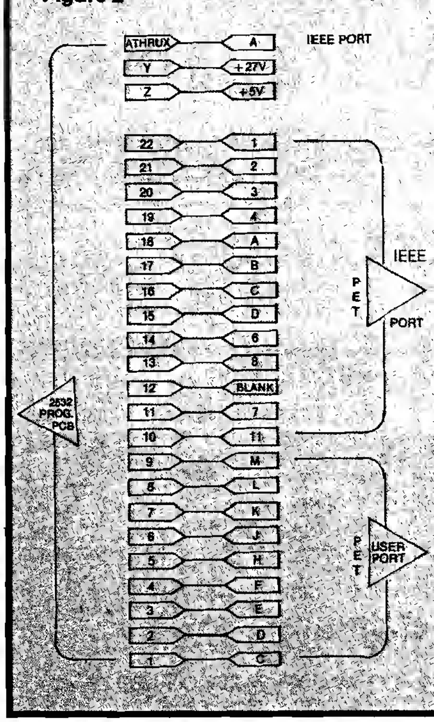
The Relocator will ask where you want the relocated program to reside. The ROM address will be the one you select; the RAM address will be within the 4K programming block of the 2532 Programmer. For example, if you want to locate your program in the top 2K of ROM 47104 ($B800) to 49151 ($BFFF), when asked for ROM address enter 47104; when asked for RAM address enter 10240. Since the ROM address is in the upper 2K of the block, the upper 2K of the 2532 Programmer programming area must be used.

After running the program the PET will display the area in ROM and RAM that the program will occupy. It will also indicate the starting point for the next program. Last but not least, it will indicate two pointer values — the BASIC START pointers, which reside in zero-page locations 40-41 ($28-$29). These are the values that you must POKE into the PET to call your ROM resident BASIC program. Write down these values for future reference.

At this point you should have a copy of the Relocator ready to be placed in ROM. Use the same procedure as described for the machine-language program to program the EPROM. When finished, instead of using a SYS command you must call the BASIC program with POKE 40,xxx and POKE 41,xxx. The xxx's equal to what was written down previously. Type RUN and RETURN. Your BASIC program will execute as if in RAM. To return to the normal PET operation you must reset the pointers with POKE 40,1:POKE 41,4.

Now, at a few keystrokes' notice, you have the TOOLKIT, SUPERMON, RABBIT, TAPE INDEX, and many other features available to enhance the operating system of your PET.

Figure 2



Jerry D. Brinson is an Electronic Engineer employed at Hueco Manufacturing, Inc. During the last four years he has been active in programming in both BASIC and assembly language. You may contact Mr. Brinson at 10136 E. 96th St., Indianapolis, IN 46256.

Listing 1: 2532 Programmer

```
100 POKE144,49:POKE59468,62:POKE59459,255:POKE59467,PEEK(59467)AND227
110 POKE59468,12:POKE53,32:POKE52,0:PRINT"J"
120 DAV=59427:NDAC=59425:AXN=59456:NRFD=59456:PROG=59468:MEM=8192
130 CB2=59467:USER=59459:ADR=59426:DXT=59471:V$="SPACE":L=0:W=1:A$="":S$=""
140 W=1:A$="":S$=""
150 INPUT"JREAD/WRITE SWITCH SET TO READ   Y";A$
160 IF A$<>"Y"THEN150
170 POKEPROG,PEEK(PROG)OR224:POKEDAV,52:POKENDAC,52:POKENRFD,PEEK(NRFD)AND253
180 POKE AXN,PEEK(AXN)AND 251:POKEADR,0
190 PRINT"J                PROGRAM MENU"
200 PRINT"P  -PROGRAM"
210 PRINT"R  -READ"
220 PRINT"C  -COMPARE"
230 PRINT"TR-TRANSFER READ DATA"
240 PRINT"TP-TRANSFER PROGRAM DATA"
250 PRINT"EM-EXIT TO MONITOR"
260 PRINT"E  -END"
270 A$="":S$="":INPUT"COMMAND ";A$
280 IFA$="P"THEN360
290 IFA$="EM"THENPOKE144,46:SYS1024
300 IFA$="R"THEN670
310 IFA$="TR"THEN880
320 IFA$="TP"THEN900
330 IFA$="E"THEN810
340 IFA$="C"THEN820
350 PRINT"TTT":GOTO 270
```

(continued)

List 1 *(continued)*

```
360 INPUT"READ/WRITE SWITCH SET TO WRITE   Y";A$:IFA$<>"Y"THEN360
370 PRINT"             ***PROGRAMMING***"
380 GOSUB1000:PRINT"   DATA FROM 8192 TO 12287":POKEPROG,PEEK(PROG)OR224:W=1
390 POKE USER,255
400 GOSUB1050:FORT1=0TO3:FORT2=0TO3:FORT3=0TO255
410 X=8192+T3+(256*T2)+(1024*T1):X1=PEEK(X):POKEDXT,X1:PRINT"DATA
420 PRINT"DATA ";X1;
430 PRINTTAB(16)" MEMORY LOCATION ";X:POKEADR,T3
440 PRINT"ADDRESS ";T3," PAGE ";T2;" SEC ",T1
450 ON (T1+1)GOSUB 610,620,630,640
460 ON(T2+1)GOSUB 570,580,590,600
470 GETS$: IF S$=CHR$(13)THENSTOP
480 POKEPROG,(PEEK(PROG)AND31)OR192:FORT=0TO25:NEXTT:POKEPROG,PEEK(PROG)OR224
490 FORT=0TO25:NEXTT
500 NEXTT3:IFJ=0THENGOSUB1050
510 NEXTT2,T1:GOTO650
520 ON (T1+1)GOSUB 610,620,630,640
530 ON(T2+1)GOSUB 570,580,590,600
540 GOSUB780
550 NEXTT3:IFJ=0THENGOSUB1050
560 NEXTT2,T1:GOTO650
570 POKEDAV,52:POKENDAC,52:RETURN
580 POKEDAV,60:RETURN
590 POKEDAV,52:POKENDAC,60:RETURN
600 POKEDAV,60:RETURN
610 GOSUB1080:POKE AXN,(PEEK(AXN)AND251):RETURN
620 GOSUB1070:POKE AXN,(PEEK(AXN)AND251):RETURN
630 GOSUB1080:POKE AXN,(PEEK(AXN)AND 251)OR4:RETURN
640 GOSUB1070:POKE AXN,(PEEK(AXN)AND 251)OR4:RETURN
650 L=L+1:IFL=LOORW=0THENPRINT"                    FINISHED":GOSUB930:GOTO140
660 IFW=1THEN400
670 W=0:IFJ=1THEN730
680 INPUT"READ/WRITE SWITCH SET TO READ   Y";A$:IFA$<>"Y"THEN680
690 POKEPROG,(PEEK(PROG)AND31)OR192
700 J=1:PRINT"          ***READING 3532 EPROM***"
710 GOSUB1000:PRINT" DATA INTO MEMORY 12288-16383
720 POKEUSER,0:PRINT"DATA  ADDRESS  PAGE  SECTOR  READ MEMORY"
730 PRINT"                                  "
740 FORT1=0TO3:FORT2=0TO3:FORT3=0TO255
750 X=12288+T3+(256*T2)+(1024*T1)
760 GETSE$:IFSE$=CHR$(13)THENSTOP
770 POKEADR,T3:GOTO520
780 X1=PEEK(DXT):POKEX,X1
790 PRINT"                                            "
800 PRINT""PEEK(DXT)"  ";T3;"    ";T2;"    ";T1;"     ";X:RETURN
810 POKE53,64:POKE52,0:POKE144,46:POKE59458,30:END
820 PRINT"";TAB(9)"PRESS RETURN TO EXIT"
830 GOSUB980:PRINT"               ***COMPARING***"
840 FORT=8192TO12287:Z=PEEK(T):Y=PEEK(T+4096)
850 IFZ<>YTHENGOSUB1020
860 IFS$=CHR$(13)THEN190
870 NEXTT:GOSUB1000:PRINT"       COMPARE COMPLETE":GOSUB930:GOTO190
880 GOSUB970:PRINT"       ***TRANSFERING READ DATA***"
890 FORT=12288TO16383:POKE(T-4096),PEEK(T):NEXT:GOTO1010
900 GOSUB970:PRINT"       ***TRANSFERING PROGRAM DATA***"
910 POKEPROG,PEEK(PROG)OR224
920 FORT=8192TO12287:POKE(T+4096),PEEK(T):NEXT:GOTO1010
930 PRINT"        PRESS "+Y$+" TO CONTINUE"
940 GETS$:IFS$=""THEN940
950 IFA$="R"THEN J=0
960 RETURN
970 PRINT"":RETURN
980 PRINT"":RETURN
990 PRINT"":RETURN
1000 PRINT"";:RETURN
1010 GOSUB1000:PRINT"       TRANSFER COMPLETE":GOSUB930:GOTO190
1020 GOSUB990:GOSUB1000:PRINT"***"T"DOES NOT ="(T+4096)"***"
1030 GOSUB1040:GOSUB930:RETURN
1040 PRINT"      PRESS RETURN TO EXIT COMMAND":RETURN
1050 PRINT"                  "
1060 PRINT"                  ":RETURN
1070 POKE NRFD,(PEEK(NRFD)AND 253)OR2:RETURN
1080 POKE NRFD,PEEK(NRFD)AND 253:RETURN
```

***Listing 2:* Relocator**

```
RELOCATOR

 0 REM RELOCATOR BY JERRY BRINSON
 2 CLR:M%=PEEK(53)/2:POKE53,M%:F=FRE(0):BE=1024:PG=256
 4 INPUT"ROM BEGIN ADDRESS IN DECIMAL";RA
 6 INPUT"MODULE BEGIN ADDRESS IN DECIMAL";MA:IFMA<M%*256THEN6
 8 O=MA-BE:IN=RA/PG:NH%=IN:RM=((IN-NH%)*256):NL%=RM
 10 IN=MA/PG:SH%=IN:RM=((IN-SH%)*PG):SL%=RM
 12 FORT=BETO(((PEEK(53)*PG)-F)-7):POKET+O,PEEK(T):PRINT"WORKING":NEXTT:PRINT
 14 LL=PEEK(MA+1):LH=PEEK(MA+2):A=(PG*SH%)+SL%+1:EN=O+(BE+((O-1)-F)-7):GOTO18
 16 LL=PEEK(A):LH=PEEK(A+1):IFLL=0ANDLH=0THEN24
 18 Z=LL+NL%:IFZ>255THENPOKEA,(Z-PG):POKEA+1,PEEK(A+1)+(NH%-3):GOTO22
 20 POKEA,PEEK(A)+NL%:POKEA+1,PEEK(A+1)+(NH%-4)
 22 A=(PG*LH)+LL+O:PRINT"":GOTO16
 24 PRINT:PRINT"PRESS <RETURN> TO CONTINUE ";
 25 POKE167,0:GETA$:POKE167,1:IFA$=""THEN25
 26 PRINT"MODULE BEGIN ADDRESS =";MA:PRINT"MODULE END ADDRESS =";A+1
 28 PRINT"BEGIN NEXT MODULE =";A+2
 30 PRINT"ROM LOCATION =";RA+((A+2)-O)-BE:POKE53,2*M%
 32 IFNL%=255THENNL%=-1:NH%=NH%+1
 34 PRINT"TO RUN BASIC MODULE":PRINT"POKE 40,";NL%+1:PRINT"POKE 41,";NH%
```

FOR YOUR APPLE II

Industry standard products at super saver discount prices

SOFTWARE

	List	SGC
ARTSCI		
Magicalc	$149.00	$ 99.00
Magic Window II	149.00	99.00
DBase (Apple)	695.00	475.00
BRODERBUND		
Payroll	$395.00	$295.00
Choplifter	34.95	25.00
Arcade Machine	44.95	29.95
Serpentine	34.95	25.00
Home Accountant	74.95	55.00
Home Accountant Plus	150.00	109.00
DATAMOST		
Snackattack	$ 29.95	$ 22.50
Thief	29.95	22.50
Swashbuckler	34.95	24.95
Zork I, II, or III	39.95	27.95
Starcross	39.95	27.95
Format II	250.00	175.00
System Saver & Fan	59.95	49.00
Multiplan	275.00	199.00
ON LINE		
Mystery House	$ 24.95	$ 19.95
Cranston Mannor	34.95	24.95
Frogger	34.95	24.95
Screen Writer II	129.95	99.95
Memory Management II	49.95	39.95
PEACHTREE		
GL, AR, AP, Inventory, Payroll	ea. $400.00	ea. $295.00
Micro Buffer II	299.00	249.00
SENSIBLE SOFTWARE		
Super Disk Copy III	$ 29.95	$ 22.95
DOS Plus	24.95	17.95
SERIUS SOFTWARE		
Bandits	$ 34.95	$ 26.95
Epoch	34.95	26.95
Fly Wars	29.95	22.95
Gorgon	39.95	29.95
Sneakers	29.95	22.95
Joy Port	74.95	59.95
Wizardry	49.95	34.95
Night of Diamonds	34.95	26.95
Star Maze	34.95	26.95
PFS	125.00	89.95
PFS Report	95.00	69.95
PFS Graph	125.00	89.95
Data Capture 4.0	64.95	49.95
Merlin/Assembly Lines	119.95	89.95
Merlin	64.95	49.95
ASCII Express Pro	129.95	99.95
Transend II	149.00	119.00
Transend I	89.00	65.00
DB Master	229.00	165.00
DB Utility Pack	99.00	79.00
STRATIGIC SIMULATION		
All Software	$ 59.95	$ 39.95
All Software	39.95	27.95
SYNERGISTIC SOFTWARE		
Wilderness & Dungeon	$ 32.50	$ 24.95
GPLE	64.95	49.95
TG Joystick	59.95	45.00
Select-A-Port	59.95	45.00
Wordstar		325.00
Spellstar		175.00
SuperCalc		175.00
VisiCalc		179.00

SPECIAL AND NEW

FRANKLIN ACE 1000 COMPUTER

Hardware and Software compatible with Apple II **$950**

FRANKLIN ACE 1000 COMPUTER plus DISK DRIVE, CONTROLLER, and MAGICALC **$1,250**

EXPAND-A-RAM™ PLUS MAGICALC™

Everything that Visicalc™ can do and much more —plus additional memory. Fully compatible with Visicalc. Includes DOS, CP/M, Pascal Disk Emulator. No preboot or Apple modification required.

64K EXPAND-A-RAM plus MAGICALC **$375**

128K EXPAND-A-RAM plus MAGICALC **$449**

APPLEsurance II™

Diagnostic Disk Controller and System Assurance Package. Standard disk controller plus automatic check of system hardware **$99**

5¼" DISK DRIVE

Use with either standard Apple II disk drive or APPLEsurance II **$249**

GRAPHITTI CARD

Prints HIRES page 1 or 2 from onboard firmware. Features: True 1:1 aspect ratio, prints emphasized mode, reverse mode, rotates 90 degrees . . . plus more. Compare all this with the Grappler. We think you'll agree that this is the best graphics card on the market. Specify for use with EPSON, NEC-8023, C-ITOH Prowriter, or Okidata.

(List: $125) **$89**

PARALLEL PRINTERS

NEC 8023 or C-ITOH 8510

(Virtually identical) Specifications: • 100 CPS dot matrix printer • 80 column print—136 characters per line • Tractor/friction feed • 7 different print fonts included • 2K printer buffer • Proportional spacing • Bit image graphics and graphic symbols.

NEC 8023 or C-ITOH **$475**

NEC 8023 or C-ITOH 8510 with Parallel Interface and Cable **$550**

EPSON 100 with Parallel Interface and Cable **$775**

BROTHER Daisywheel Printer **$895**

VERSAcard FROM PROMETHEUS

Four cards on one! With true simultaneous operation. Includes: (1) Serial Input/Output Interface, (2) Parallel Output Interface, (3) Precision Clock/Calendar, and (4) BSR Control. All on one card. Fully compatible with CP/M* and Apple Pascal*.

(List: $249) **$169**

WORD PROCESSING SPECIAL WITH WORDSTAR AND SUPERCALC!

Do professional word processing on your APPLE. All necessary hardware and software included. Complete 80 column video display, enhanced character set, 16K memory board, Z-Card with CP/M* software, Wordstar and word processing software and SuperCALC.

(List: $1,228) . . . **Special at $795**

	List	SGC
Z-80 CARDS		
Microsoft Softcard Z-80	$399.00	$289.00
ALS Synergizer	749.00	595.00
U-Z-80 Processor Board		125.00
Microsoft + Premium Syst.		595.00
80-COLUMN CARDS		
Smarterm 80-Col Display	$345.00	$225.00
Smarterm Expanded Character Set		40.00
Combination Smarterm & Exp. Char. Set		260.00
Videx Videoterm		275.00
Videx Enhanacer II	149.00	125.00
Videx VisiCalc Preboot	49.95	45.00
MODEMS FOR YOUR APPLE II		
Hayes Smartmodem 300		$229.00
Hayes Smartmodem 1200	699.00	550.00
Micromodem II		279.00
Hayes 100 Baud		Call
Apple Cat II	389.00	299.00
D Cat Modem	199.00	175.00
MONITORS		
Amdek 300G Green		$159.00
Color—Taxam RGB with Interface		395.00
PARALLEL INTERFACE		
Centronics Compat. PRT-1		$ 69.00
JOYSTICK Replaces two Apple Paddle Controllers	$ 59.00	$ 39.00
FUNCTION STRIP	$ 79.00	$ 65.00
MEMORY EXPANSION		
Prometheus 16K RAM Module complete	$169.00	$ 65.00
5¼" FLOPPY DISKS		
Box of 10 with hub rings		
With other purchase		$ 19.95
Without other purchase		23.00

All equipment shipped factory fresh. Manufacturers' warranties included. California customers add 6½% tax. Include payment by personal check, money order, or cashier's check with order and SGC will pay shipping charge. Call for amount of shipping charge when paying by credit card.

All items are normally in stock

(415) 490-3420

. . . And we'll be here to help after you receive your order. Feel free to call the SGC Technical Staff for assistance.

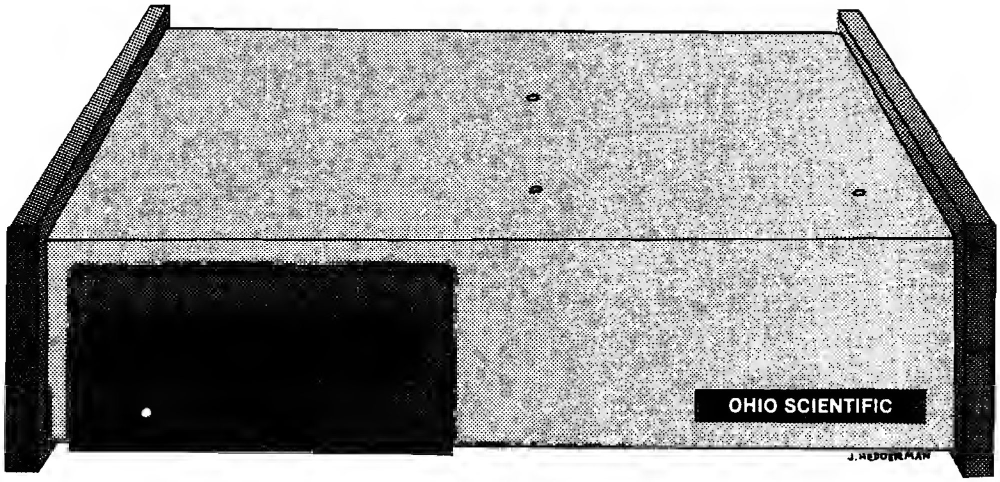
SGC

The mail order specialists

342 Quartz Circle, Livermore, CA 94550

Circle No. 36

AUTOMATIC HEAD LOAD CONTROL



by Peter Kleijnjan

By adding just a few bytes of patch code to the operating system, OSI minifloppy system users can have fancy head load control. Now the head is loaded only during disk activity.

OSI minifloppy system users may have noticed the LED that glows continually on the front of their disk drives. This LED is on whenever the disk head magnet is activated and the head is pulled down to touch the surface of the diskette: so-called head load. Whenever your computer system is on, the disk head is loaded, and the glowing LED is a silent witness to the wearing away of your disk head and floppies. In addition, that handsome auto-eject mechanism of your floppy drives won't work when you keep the head loaded all the time.

There is an elegant way to reduce diskette and disk head wear: just unload the disk head when there is no disk activity. This type of head load control is a standard feature of nearly all professional floppy disk-based systems, including OSI's C3 and C8P 8'' systems. OS-65D was originally developed for use on these 8'' systems, and this explains why most of the software you need to automatically load and unload the disk head is already part of it. All OS-65D commands that require disk access are tied into head loading and unloading routines. These routines, however, were not written to control the MPI B51, the disk drive used in OSI's minifloppy systems.

The minifloppy disk drive controller is connected to a PIA at $C002, with separate bits for stepping, reading/writing, erasing, track 0 indication, etc. OS-65D drives a head load bit at D7 of the PIA, but this bit is not connected to anything meaningful on minifloppy systems. The MPI drive controller does not have a separate head load control input. However, a drive select control input at D5 that is used by the operating system's SElect command, unloads the head when it is off select. Now, if you don't use more than one minifloppy drive, there is no reason why you shouldn't use this input to control your head load.

The first thing you have to do is convince the operating system that the head load bit is at D5 (counting D0-D7), active high, instead of D7, active low. This is easy: instead of resetting D7 to load the head to disk, set D5, and *vice versa*. There is a little more to it than this, however. The operating system *first* positions the head and *then* loads it to disk, which is perfectly logical. You are using the drive select, however, instead of the nonexistent head load. This means that when the operating system tries to position the head, the drive is not listening (it's off select) and you get read errors. What you have to do is to call your load-head routine *before* you position the head. This is achieved by installing patches (PATCH1 and PATCH2) in both the position-head-to-track and the HOme routine.

When you look at the code, you'll notice a few quirks that require explanation. OS-65D V3.2 has an undocumented bonus: sector 2 on track 6 contains extensions to the operating system that are loaded to $3200 for one

Listing 1: **Head Load Control Assembly Listing**

```
                     ;MODIFICATION TO USE THE AUTOMATIC
                     ;HEAD LOAD ON/OFF FEATURE OF OS65D
                     ;V 3.2 ON C2-4P AND C4P SYSTEMS
                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                             ;
                             ;LOAD HEAD---
                             ;SET BIT 5 AT $C002
                             ;
    2754                     *=$2754
    2754 A920     LOAD       LDA #$20
    2756 0D02C0              ORA $C002
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;
                             ;
                             ;UNLOAD HEAD---
                             ;RESET BIT 5 AT $C002
                             ;
    2761                     *=$2761
    2761 A9DF     UNLOAD     LDA #$DF
    2763 2D02C0              AND $C002
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                             ;
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    2663                     *=$2663 ;(HOME)
    2663 202432              JSR PATCH2
    2E17                     *=$2E17 ;(FIND NAME
                             IN DIR)
    2E17 202B32              JSR PATCH4
    2294                     *=$2294
    2294 4CA922              JMP $22A9
    22A9                     *=$22A9
    22A9 A91E                LDA #$1E
    22AB 8DD826              STA $26D8
    22AE A932                LDA #$32
    22B0 8DD926              STA $26D9
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                             ;
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    321E                     *=$321E
    321E 205427   PATCH1     JSR $2754
    3221 4CDA29              JMP $29DA
    3224 205427   PATCH2     JSR $2754
    3227 208A26              JSR $268A
    322A 60                  RTS
    322B A960     PATCH4     LDA #$60   ;SELF-MOD CODE
    322D 8D202B              STA $2B20       ;@ $2B20
    3230 201A2B              JSR $2B1A
    3233 A94C                LDA #$4C
    3235 8D202B              STA $2B20
    3238 60                  RTS
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                             ;
    27C1                     *=$27C1
    27C1 94                  .BYT $94
```

Head Load Control
requires:
C2/4P or C4P with OS65D V3.2

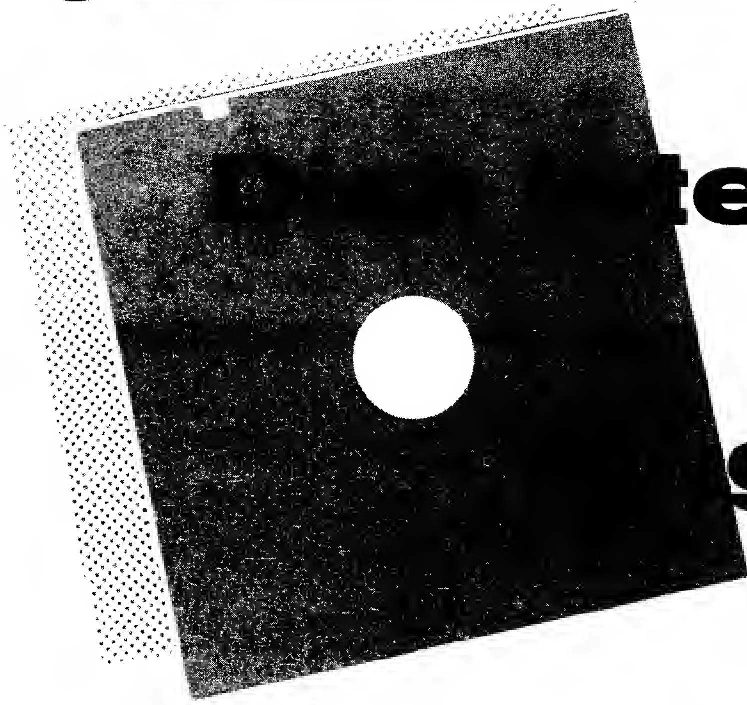
page. In this page, OSI has placed a modified keyboard routine instead of the one in ROM at $FD00. Although this routine is just as bizarre as the old one, you can appreciate the gesture and use the space that is left over to accommodate your patches. The keyboard routine goes from $3200 to $321D, so your patches start at $321E. Since the patches reside on track 6,2 you cannot call them before the operating system is loaded in full. Therefore, install the final patch in the position-head-to-track routine only just before you start loading BASIC.

After these changes, there are two commands that do not unload the head upon return: HOme and INitialize. In both cases, the disk head is automatically unloaded again on completion of the next disk-accessing command. Listing 2 is a BASIC program to install the modifications. I must emphasize that *only* V3.2 can be used: the code lay-out and timing of V3.0 and V3.1 are totally different.

You may contact Mr. Kleijnjan at Kleijnjan Consultants BV, Kerkwetering 11, 3421 TS Oudewater, The Netherlands.

Listing 2: **Head Load Control BASIC Listing**

```
100 PRINT:PRINT"Make sure that you have a OS-65D V3.2 system disk"
110 PRINT"in the disk drive with track 0 copier on 13,1;"
120 PRINT"then press any key.":DISK!"GO 252B"
130 DISK!"CA 5200=06,2"
140 FORI=0TO26:READA:POKE21022+I,A:NEXT
150 DISK!"SA 06,2=5200/1"
160 DISK!"CA 4A00=01,1"
170 POKE19991,32:POKE19992,43:POKE19993,50
180 DISK!"SA 01,1=4A00/8"
190 POKE9105,0:POKE9106,80:REM SET #5 POINTERS
200 LF$=CHR$(10):CR$=CHR$(13):CA$=CHR$(64)
210 PRINT#5,"EXIT"+CR$+"CA 0200=13,1"+CR$;
220 PRINT#5,"GO 0200"+CR$+"2"+CR$+"R4200"+CR$;
230 PRINT#5,"E"+CR$+"EM"+CR$;:PRINT#9
240 PRINT#5,CA$+"4754"+CR$+"A9"+LF$+"20"+LF$+"0D"+LF$;
250 PRINT#5,"02"+CR$;:PRINT#9
260 PRINT#5,CA$+"4761"+CR$+"A9"+LF$+"DF"+LF$+"2D"+LF$;
270 PRINT#5,"02"+CR$;:PRINT#9
280 PRINT#5,CA$+"4663"+CR$+"20"+LF$+"24"+LF$+"32"+CR$;:PRINT#9
290 PRINT#5,CA$+"4294"+CR$+"4C"+LF$+"A9"+LF$+"22"+CR$;:PRINT#9
300 PRINT#5,CA$+"42A9"+CR$+"A9"+LF$+"1E"+LF$+"8D"+LF$+"D8";
310 PRINT#5,LF$+"26"+LF$+"A9"+LF$+"32"+LF$+"8D"+LF$+"D9";
320 PRINT#5,LF$+"26"+CR$+CA$+"47C1"+CR$+"94"+CR$;:PRINT#9
330 PRINT#5,"EX"+CR$+"CA 0200=13,1"+CR$+"GO 0200"+CR$;
340 PRINT#5,"2"+CR$+"W4200/2200,8"+CR$+"E"+CR$+"GO FFA0"+CR$
350 POKE9098,0:POKE9099,80:DISK!"IO 10,02":END
360 DATA32,84,39,76,218,41,32,84,39,32,138,38,96
370 DATA169,96,141,32,43,32,26,43,169,76,141,32,43,96
```



erface for gle Board 6502s

by Jack Brindle

Complete hardware diagrams and sample software drivers help explain how to interface a floppy disk to a 6502-based system.

Many single-board computers have cassette interfaces built into the system. These interfaces provide adequate, low-cost mass storage for beginning programmers or small system jobs. However, as you progress in computing, the job of editing, assembling (or compiling), and loading becomes tedious due to the slow data transfer rates inherent with tape systems.

For several years floppy disks have been the standard mass storage device for development systems. Until recently these systems were too expensive for those of us on an experimenter's budget. Floppy controllers generally were large boards containing many diverse components, and required special circuitry to interface to a microcomputer system.

Western Digital simplified the design with the introduction of the FDD1771 floppy disk controller. This chip placed the tasks of data accumulation, timing, block check control and drive control into one LSI package. A second generation chip, the FDD1791 expanded upon the FDD1771, while solving processor interface and data detection problems. Western Digital now manufactures a family of disk controllers, the FDD179X series of devices. Each member of the family offers variations of the FDD1791, such as a true or inverting data bus, and double-density capability. Another chip recently introduced by Western Digital, the WD1691, handles the separation and detection of data from the floppy drive through a phase-lock loop arrangement. These devices form the basis for an inexpensive mass storage system for the small computer.

The standard recording technique used in most floppy disk systems is the IBM 3740 format. The FDD179X controllers use this format for data storage, allowing easy interchange of data between users. The 3740 format uses 128-byte sectors with 26 sectors per track and 77 tracks per drive. The 77 tracks recorded on each floppy contain a lot of information (see figure 1). In addition to the 26 data fields, there are pre-index fields, post-index fields, gap fields, and other fields used by the controller to keep track of exactly what data it is reading. The track and sector identifiers are recorded to provide a check to the controller for its internal count registers. Each sector contains error detection information in the form of cyclic redundancy check (CRC) characters. The controller compares these characters with figures it has accumulated during the data transfer. Any differences are flagged as an error by the controller.

I chose the FDD1793 for my system because of its double-density capability and true data bus. The addition of 12 integrated circuits form the basis for the floppy system. I use an eight-inch Siemens FDD-100-8A floppy disk drive as the mass storage device, but a mini-floppy could just as easily be interfaced

with one minor change. Appropriate control software finishes off the system, allowing the main processor to control all data transfers.

The target computer for the floppy storage system consists of an MCS6502 microprocessor with a clock speed of 1.25 MHz, 28 kilobytes of static RAM, and 12 kilobytes of EPROM. Previously, I used a cassette interface for mass storage. This system provides the basis for a powerful computing system.

Hardware

See figure 2 for a schematic of the floppy disk interface. The system can be broken into three parts. U1, 2, 8, and 10 comprise the bus interface to properly connect the system's components to the OSI bus. U1 and 2 are four-bit bidirectional buffers used to invert the normal data for the system data bus. The direction of data travel in these buffers is determined by U10, a fast PROM used to decode the address bus, enabling the on-board devices when addressed. I decided to use a TMS2508 at U10 during development because of programming ease. This device can be programmed using a 2716-type EPROM programmer. I later changed the device to an MMI 6341 bipolar PROM. This eliminates timing problems caused by the slower EPROM when you're accessing memory devices. The PROM is programmed to provide a low output on a device-select output line only when the desired device address is applied to the address lines. Bit 7 of the PROM provides data direction information, so it must be programmed to output a zero for a data read of a selected device. Figure 3 gives examples of the PROM data. By using a PROM you are allowed flexibility in the addresses used for selecting devices on board. The unused output lines of the decoder can be used to select other devices on board, such as parallel or serial I/O interfaces.

The RD and WR signals required by the FDD1793 are derived from the system phase 2 clock and R/W signal by U8, a 74LS00 TTL quad NAND gate.

The second section of the system consists of the disk controller and its associated circuitry. I chose an FDD1793 for its normal (non-inverted) data bus and double-density format capability. The FDC requires a 2MHz symmetrical square wave at its clock input. This signal is derived from a 4 MHz crystal-controlled clock generated by two sections of U7. A 74LS04 should not be used here to avoid startup problems created by the 74LS04's low input impedance. The 4 MHz clock is divided by two by one section of U11, a 74LS74 D-type flip-flop. The output is a 2 MHz square wave used to drive the FDD1793. For use with a minifloppy, the signal should again be divided by two to give a 1 MHz square wave required for the slower minifloppy data rate.

The task of data separation and recovery is handled by U5 and 6. U5 is a Western Digital WD1691 Floppy Support Logic device. It works in conjunction with U6, a 74S124 voltage-controlled oscillator, as a phase-lock loop (PLL) data separator. The FM encoded data recorded on the diskette consists of clock and data pulses. Each data cell begins with a clock pulse. If the data bit is a '1', a data pulse follows in the middle of the cell; no pulse is recorded for a '0'. The WD1691 senses each clock pulse, transmitting it to the FDC on detection. The frequency of the clock pulses is monitored by the PLL, causing the PLL frequency to follow the input rate. Only the clock pulses need to be separated; the FDC takes care of separating the data pulses from the input stream.

U12 provides a delay for the disk controller to assure that the disk read/write head is fully loaded prior to attempting any read or write with the disk. The period of the one-shot, which is about 33 milliseconds for a Siemens FDD-100-8A, can be adjusted by changing R3 to meet the requirements of other drives. Signals going from the controller board to the floppy disk drive are buffered with 7406 inverting open collector buffers. This provides the inverted signal levels required by the disk drive. Input signals are generally unbuffered, although the Raw-Read and Ready signals are both buffered to reduce noise pickup. All lines use one kilohm pull-up resistors as terminators to reduce noise and provide proper TTL level signals. The cable connecting the board to the floppy drive should be kept as short as possible.

Although based on the OSI-48 bus, the interface could easily be adapted, with minor software modifications, to other systems.

The third part of the system is used to lessen the software requirements to drive the FDC. This section consists of an MCS6522 Versatile Interface Adapter (VIA). The FDC's DRQ and IRQ lines are routed to PA7 and PA6 of the VIA. This allows a single 6502 'BIT' instruction to read the status of the FDC, saving valuable time in interrogating the FDC for data. Bit 0 of port A is used to select the disk drive in a two-drive system. The remaining bits are available for other control or status uses as desired. Additionally, a full eight-bit port with handshaking is available for

(Continued on page 59)

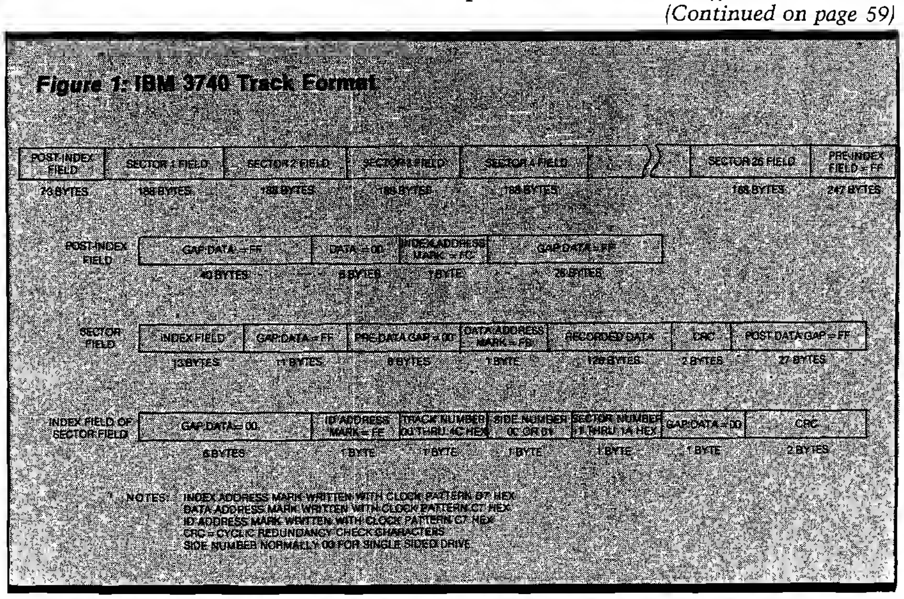
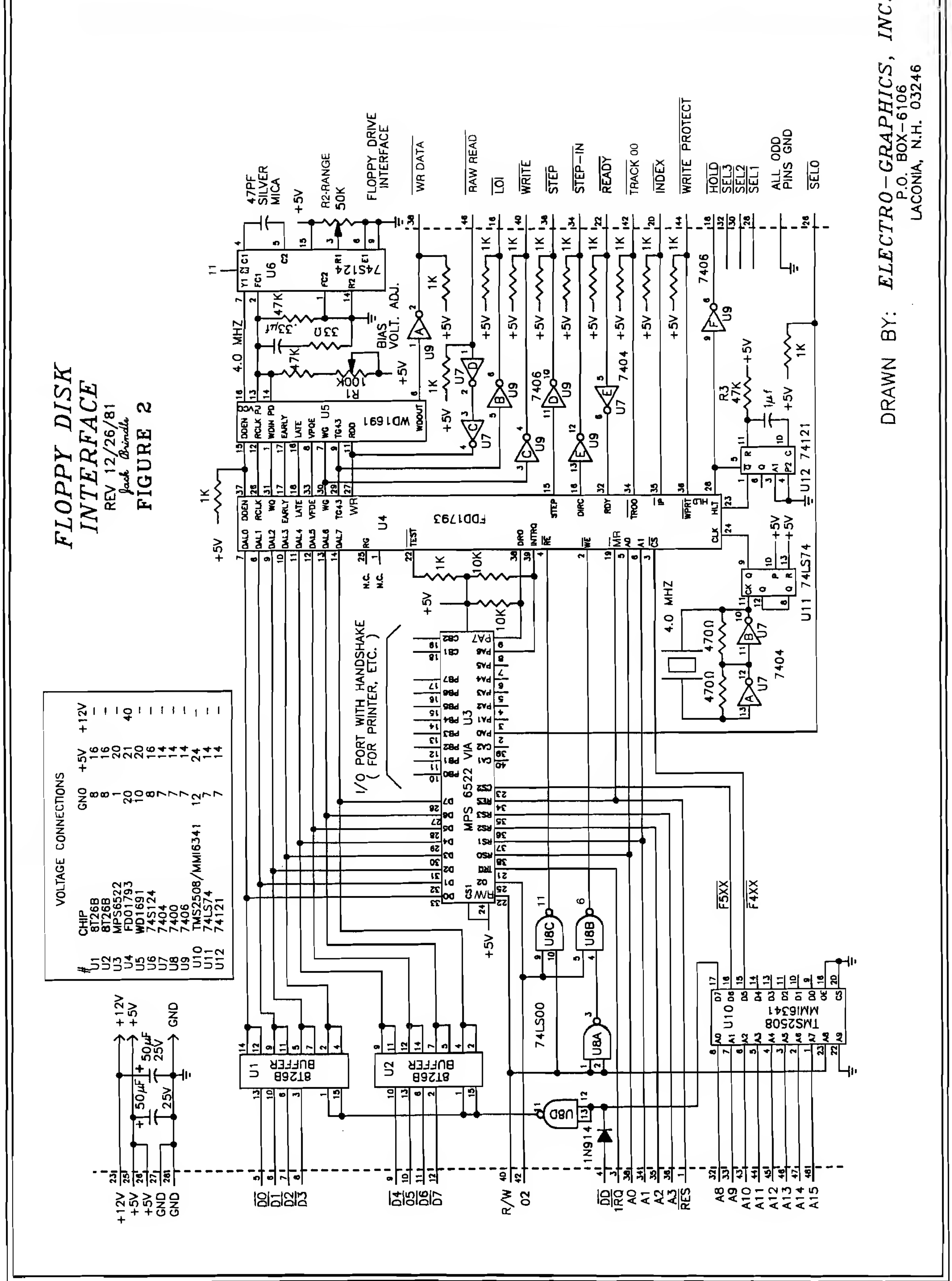


Figure 1: IBM 3740 Track Format

Figure 2: Interface Schematic



driving a printer or other I/O device.

The parts for this project, available from several parts distributers advertising in computer magazines, cost about $90.00. The FDD1793 is the most expensive part.

Adjustment

To adjust the PLL made up by the WD1691 and 74LS124 use the following steps: disconnect the disk drive from the circuit. Connect the Raw-Read line to +5 volts. Adjust R1 (Bias Voltage Adjust) for a reading of 1.4 volts at pin 2 of U6. You should do this with a high-impedance voltmeter to avoid loading the circuit. Next connect a frequency counter to pin 7 of the 74S124. Adjust R2 to give a 4 MHz signal at this point. These are the only adjustments required for the system to operate normally.

Software

The software that drives the floppy disk controller is written using a multi-level approach. The fundamental level contains the disk control and read/write routines that interface directly to the controller. The second level builds on these routines to provide a file handling structure. User programs rely on both levels to support their execution.

You achieve disk control by issuing commands to the FDD1793 (FDC). These commands are broken into four levels (see figure 4). The first level controls the position of the read/write head over the diskette. Level two is used for normal sector data transfers and level three is used generally only for formatting the diskette. Level four gives a facility for interrupting disk functions and placing the FDC in a known state prior to issuing a disk command.

The controller status is available in an eight-bit read-only reigster. The contents and meaning of the status register is dependent on the currently executing command, or that most recently executed. A summary of the bit designations for type 1, read sector and write sector commands is given in figure 5. Upon termination of a command, the register is tested for proper execution. If an error flag is set, use an error procedure to recover the invalid command execution.

On reset the Force Interrupt command is issued, using its terminate function to place the controller in a ready mode, then a home command restores the read/write head to track 00. This process clears the FDD1793's internal track counter to 0, with the FDC ready to accept further commands. The controller can now be commanded to read or write data, or move to a desired track to get data. To move to another track, the user can issue multiple step, step-in, or step-out commands until the head is positioned over the desired track. An easier way to accomplish this task is to load the data register with the desired track and issue a seek command. This allows the controller to perform the necessary steps for the CPU, signalling an IRQ when finished. This command is illustrated in listing 1.

Reading and writing data to the diskette is a simple task when done with the FDC. As listing 2 shows, the desired sector ID is loaded into the sector register, then the proper read or write command is issued. A wait loop is then executed, waiting for the DRQ line to become active. The DRQ bit in the FDC's status register could be tested, but a longer operation would be necessary since the data must be ANDed with a bit mask to get the proper bit. Further, the busy flag information would be lost, requiring more processing to recover the data. By using the 6522, these extra steps are eliminated, saving both time and memory space. The IRQ bit is tested first to assure that the operation has not come to an error halt. The DRQ bit is tested until it becomes active and the data byte is read and stored. The location counter is updated; the operation continues until 128 bytes have been transferred or an error interrupt occurs. Finally the status register is read and checked for

Figure 3: [illegible]

DEVICE	[illegible] ADDRESS	[illegible] BIT	PROM ADDRESS	DATA
READ FDC	[illegible]	[illegible]	[illegible]	[illegible]
WRITE FDC	[illegible]	[illegible]	[illegible]	[illegible]
READ VIA	[illegible]	[illegible]	[illegible]	[illegible]
WRITE VIA	[illegible]	[illegible]	[illegible]	[illegible]

All other locations [illegible] FF.

Note that only the bit for the selected [illegible] low. [illegible] low for the read of the selected device, indicating that data will travel [illegible] the CPU [illegible].

Figure 4: FDC Commands

1 RESTORE	0000hvab	h = head load flag
1 SEEK	0001hvab	1 = load, 0 = unload
1 STEP	001uhvab	v = verify flag
1 STEP-IN	010uhvab	1 = verify, 0 = no
1 STEP-OUT	011uhvab	a b = stepping rates
		00 = 3 ms
		01 = 6 ms
		10 = 10 ms
		11 = 15 ms
2 READ SECTOR	100m1e00	m = multiple record flag
2 WRITE SECTOR	101m1e0c	1 = multiple records
		e = 15 ms delay
		c = 0 = data addr mark
		1 = data deleted mark
3 READ ADDRESS	11000e00	
3 READ TRACK	11100e00	
3 WRITE TRACK	11110e00	
4 FORCE IRQ	1101ijkl	i = immediate interrupt
		j index pulse
		k = rdy to not rdy transition
		l = not rdy to rdy transition
		all 0 = terminate, no IRQ

proper execution. If an error occurred, the operating system handles this by retrying the operation a specified number of times. Should the error continue, the command is aborted and the operator notified. Note that writing to the disk is performed in the same way, with the data transfer going to the FDC from the memory buffer.

Now I will turn to the second level of the software system. As discussed previously, this level uses the basic sector read, write, and disk control functions from the first level to provide a file-handling structure. Rather than write this level on my own and be saddled with a one-of-a-kind system, I decided to conduct a search for a commerically available operating system to run on my system. The 6502 user has had a major disadvantage compared to other processor users with the lack of a common operating system. This has made software exchange next to impossible with so many standards

(Continued on page 62)

Listing 1: SEEK Example

```
FCMD    EQU   $F400          FDC command register (write only)
FSTTS   EQU   $F400          FDC status register (read only)
FTRCK   EQU   $F401          FDC track register
FSECT   EQU   $F402          FDC sector register
FDATA   EQU   $F403          FDC data register
VIA     EQU   $E50F          6522 VIA port 'A' data
SEEK    LDA   TRACK          get the destination track
        CMP   FTRCK          if equal to current track
        BEQ   EXIT           then leave, seek is unnecessary
        STA   FDATA          load track into FDC
        LDA   #%00010111     issue seek command
        STA   FCMD           to FDC
LOOP    BIT   VIA            get status, loop until
        BVC   LOOP           IRQ is signalled
        LDA   FSTTS          get command status for check
        AND   #%10010000     error if not zero
EXIT    RTS                  finished!
```

floating about. The availability of a quality, common operating system has helped push the popularity of Z80-based computers.

After several months of searching, I finally found my operating system. Called DOS/65 it is CP/M file-compatible, allowing files to be read and written from CP/M version 1.4 diskettes. Since we are using a 6502 processor, the CP/M programs cannot be executed, except for those written in BASIC. DOS/65 includes a good assembler with loader, editor, debugger and BASIC-E/65 BASIC compiler/interpreter. The assembler object output is in KIM file format, which can be converted to absolute executable code using the system loader. Assembly-language programs can be written on the system, assembled and debugged, then downloaded to a KIM, SYM, or TIM system, or punched to paper tape for loading by one of these systems. This feature is useful for developing programs for controllers or small systems.

(continued)

Listing 2: Sector READ Routine

```
FCMD    EQU   $F400        FDC command register (write only)
FSTTS   EQU   $F400        FDC status register (read only)
FTRCK   EQU   $F401        FDC track register
FSECT   EQU   $F402        FDC sector register
FDATA   EQU   $F403        FDC data register
VIA     EQU   $F50F        6522 VIA port 'A' data
READ    LDA   SECTOR       get the sector
        STA   FSECT        to the FDC
        LDY   #$00         zero index
        LDA   #%10001000   issue read sector command
LOOP    BIT   VIA          get DRQ, IRQ status
        BVS   DONE         error if IRQ = 1
        BPL   LOOP         wait until DRQ = 1
        LDA   FDATA        get the data byte
        STA   BUFFER,Y     store the data byte
        INY                next location
        BPL   LOOP         loop until 128 bytes input
DONE    LDA   FSTTS        get FDC status
        AND   #%10011100   error if Acc not zero
        RTS                finished!
```

Figure 5: Status Register Summary

BIT	TYPE 1 CMD	READ SECTOR	WRITE SECTOR
7	NOT READY	NOT READY	NOT READY
6	WRITE PROTECT	0	WRITE PROTECT
5	HEAD LOADED	RECORD TYPE	WRITE FAULT
4	SEEK ERROR	RECORD NOT FOUND	RECORD NOT FOUND
3	CRC ERROR	CRC ERROR	[illegible]
2	TRACK 0	LOST DATA	LOST DATA
1	INDEX	DRQ	DRQ
0	BUSY	BUSY	BUSY

DOS/65, like CP/M, is written in several layers. The first layer, the System Interface Module (SIM), includes the disk drivers discussed previously, along with system console and printer drivers. The second level, called PEM or Primitive Interface Module, links individual sectors together to form files. Sectors are dynamically allocated: as sectors are needed for file data, they are allocated from a free pool of sectors. Thus the sectors allocated to a file need not be contiguous, and generally are not. This eliminates the need for sector packing found in many systems that can cause problems if a power interruption occurs during a pack operation. The third layer of DOS/65 is called the CCM, or Console Command Monitor. This layer provides a user interface using the two lower levels. Through this interface, the user can develop, compile, and execute programs on the system without worrying about the specifics of the system hardware.

These three layers give DOS/65 much power and flexibility; I highly recommended this system to the 6502 user. (DOS/65 is availble from Micro Systems Engineering, 1363 Nathan Hale Dr., Phoenixville, PA 19460.)

Conclusion

The floppy disk system discussed in this article is a versatile yet simple system for the microcomputer user. The small component count and use of the CPU in data transfers provides a low cost system with enough power to control several floppy disk drives. The system software helps give the user a "big system" feeling. A computer using these components will provide a good tool for software development or a general purpose computer capable of meeting all but the most demanding user's needs.

I would like to thank Mark Gintis and Richard Leary for their help in this project, and Western Digital for their exceptional diskette controller.

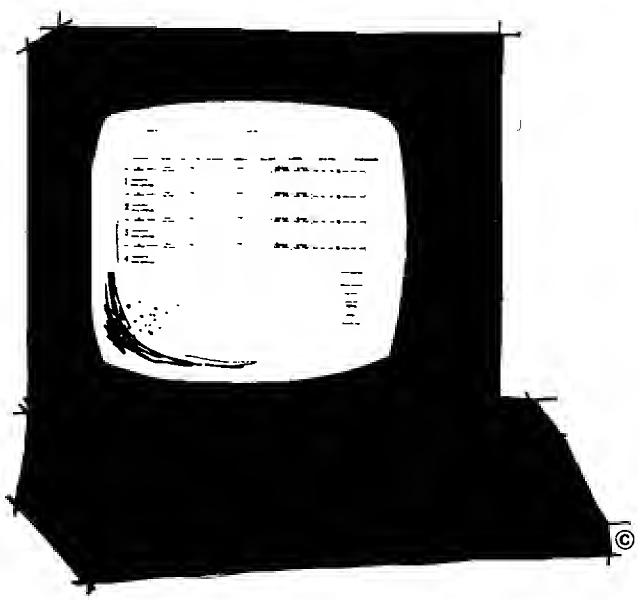
Mr. Brindle is a design engineer with Motorola's Portable Products Division. His system is a homebrew 6502-based computer, which has evolved from a single OSI 400 board to a disk-based 28K system. You may contact Mr. Brindle at 1174 N.W. 29th St., Sunrise, FL 33323.

MICRO

The MICRO™ LEARNING CENTER

TRS-80C **Texas Instruments**

Features:



- **Mode 10 Atari Painting Program**
  by Paul Swanson

  A simple drawing program that lets you use nine colors

- **Word Detective**
  by Bob Tripp

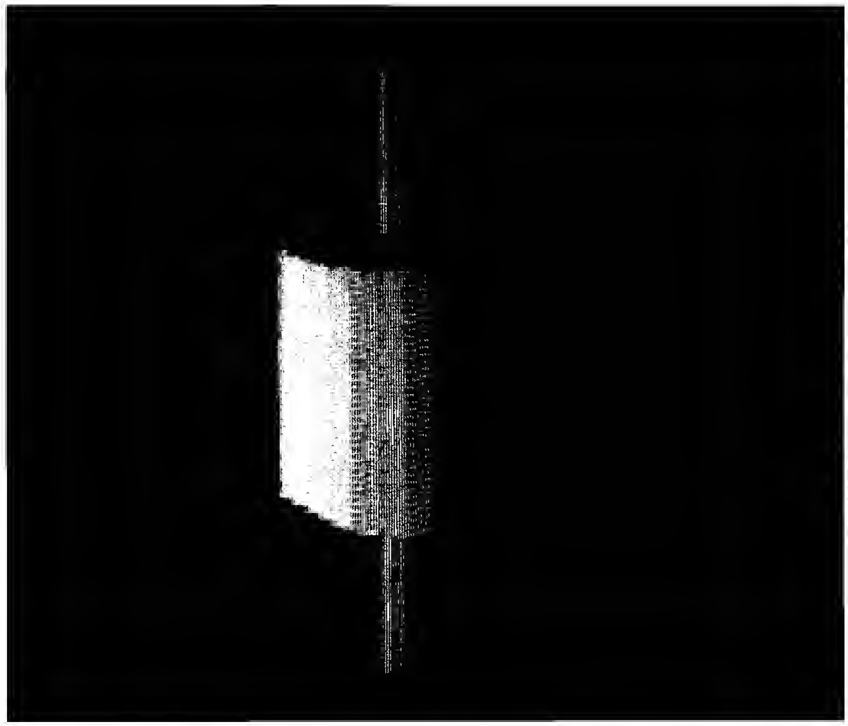
  Figure out why, in this word game, the computer accepts some words and not others

- **Lo-Res Shape Drawer**
  by Doug Denby

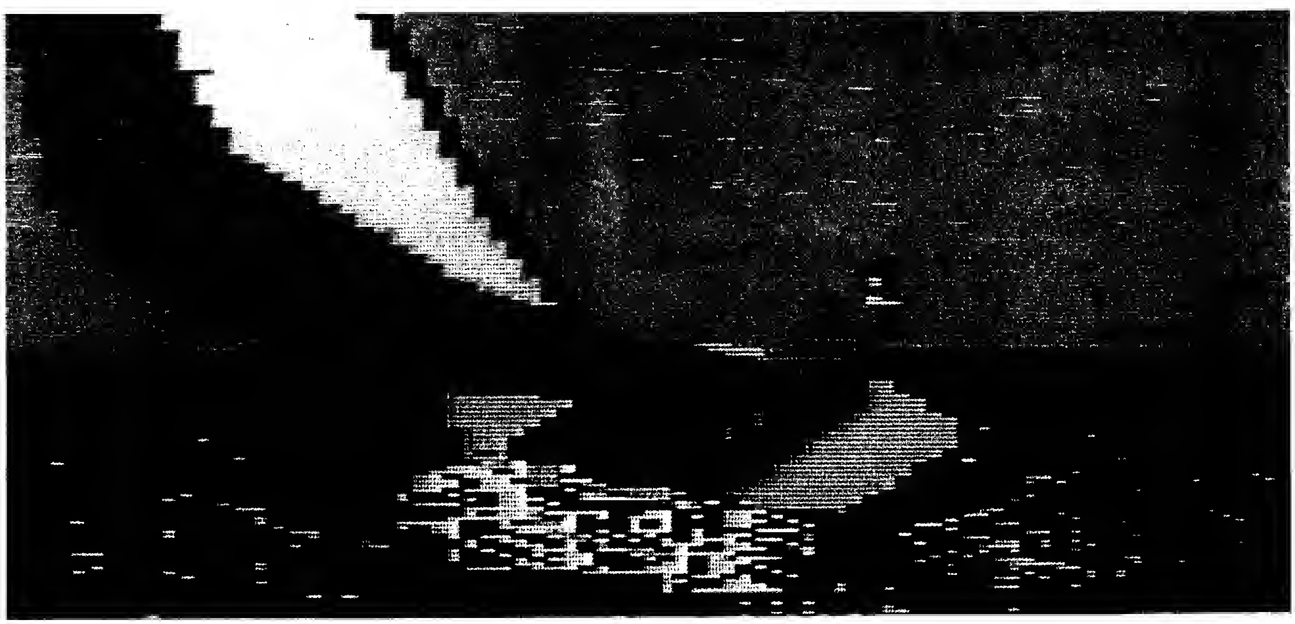
  Draw enlarged versions of hi-res shapes on the lo-res screen

Mode 10 Atari Painting Program

by Paul Swanson



The program in the accompanying listing can be entered and used to create screen displays with up to nine different colors on your screen. Included are options for loading and saving pictures on disk, and functions may be easily added to the program to expand its capabilities.



Also, the pictures stored on the diskette can be loaded and used by other BASIC programs so that you can incorporate your drawings in your programs without having to decipher all the PLOT and DRAWTO commands required.

After you enter the program, save it on cassette or diskette using either SAVE or LIST. If you alter the program substantially, LIST it to disk or cassette, enter NEW, then ENTER the program again. This clears the variable table and a few other things that make the program run more efficiently. Actually, that is good practice when altering any significantly sized program.

This month Part I explains how to operate the program so that you can type it in and know how to use it. Next month, Part II will explain in detail how the program operates and how you can add your own functions to it.

Entering the Listing

Most of the program can be entered with no explanation — just enter it as it is listed. However, there are a few lines in the program that contain control codes and control codes do odd things to printers. Therefore, the listing was made using a special lister program that converts each control code to a form that is printable — a number enclosed in angle brackets. Consult Appendix C in your *BASIC Reference Manual* for the conversion to keystrokes.

When entering control codes, you most often use CTRL and a letter key. For example, < 1 > is a CTRL-A. Note that <0 > is a CTRL-comma. These codes all have special symbols that print out on the screen, like the heart that prints out when you enter CTRL-comma.

There are a few control codes that require the ESC key. These correspond to the screen and cursor controls. For example, the up arrow lists as < 28 >; press the ESC key once, then press the CTRL-up arrow. For < 27> , just press the ESC key twice. The control codes that require the ESC key tend to complicate entering the listing. However, with a little trial and a lot of error, things will start to make sense.

Using angle brackets can create confusion since BASIC uses them for greater than and less than. If there is confusion on any angle bracket, first check to see if the brackets are enclosed in quotation marks. If they are, the number is a control code; otherwise it is an angle bracket indicating greater than or less than.

There is one screen control code that will not print in angle brackets because it isn't a code that requires the CTRL key. Instead, it appears as a right brace and is actually a clear screen code misinterpreted by the printer. In line 22 the first symbol after the quotation mark is this right brace. To enter, press ESC once, then press SHIFT with CLEAR. You should get an arrow that points up and left.

Running the Program

Once you have typed the listing in and SAVEd it, type RUN and hit RETURN. The program will begin a rather lengthy initialization routine. After 15 seconds, the entire screen will turn black except for one small, blinking dot in the center. If you have trouble seeing the dot, turn the brightness up on your television. This dot is the cursor and it marks where you are on the screen. To move the cursor, use the joystick. The cursor will move in a direction corresponding to the direction in which you push the stick. Next, hold the trigger button down while you move the joystick. If everything is typed correctly, you should be able to see that the joystick controls cursor movement and the trigger controls the depositing of color.

Mode 10

The mode 10 graphics mode has the most flexible variety of colors but it also has a strangely shaped pixel, or dot of color. The pixel in mode 10, as in the other so-called GTIA modes, is long horizontally (there are only 80 of them across the screen), but short vertically (192 of them from top to bottom). Since the timing in the program depends on the number of dots to move, rather than actual distance, horizontal movement appears to be about four times faster than vertical movement.

Although the cursor movement responds to the diagonal positions of the joystick, the resulting line does not appear at 45°. This is another effect of the odd pixel shape. The diagonal joystick position produces a line that travels four units horizontally for each unit vertically.

Other Colors

There is more to this program than just that dull reddish line on the screen! There are eight foreground colors in addition to the black background. You can chose any one of them for your "paintbrush."

Push START. A multicolored bar should appear on your screen and, after a second or two, a white arrow and the abbreviation COLR under the bar. This is the selector for your paintbrush color. It is initially pointing to the left of the colored bar, which is the position to select the background color. This can be used to erase mistakes or to "cut out" holes in solid shapes. Push the joystick right or left to position the arrow under the color you want and press the joystick trigger button. The drawing in progress reappears on the screen and you can now draw with the new color you selected.

Each of the eight colors plus the background color can be changed to any one of 128 colors. When you change any one of the colors on the bar,

all that you have drawn using that color will change to the new color. To see how this works, press SELECT. The screen looks exactly like the one to select the paintbrush color except the abbreviation under the arrow is CHGE instead of COLR. Use the joystick to place the arrow under the color you want to change and press the trigger.

A longer bar of colors will now appear on the screen. There are sixteen in all, including black for the background (black will produce all of the grey shades from black to white). For this example, move the HUE arrow to under one of the blue colors, then press the trigger button.

On the next screen, using an arrow labelled LUM, you can select the luminance, or brightness, of the hue you chose. Since you chose blue because you are following this example, the top third of the screen all around the bar should be the darkest shade of blue. Because dark blue does not show up well on a black background, try choosing a lighter blue by moving the arrow over to the eighth shade, and pressing the trigger button. Notice that all of the lines you drew in that dull red color changed to a light blue. If you select another color for the first color on the bar, all of the lines will change to the new color.

One note about the luminances concerns the differences in modes. The luminance bar appears to give you a choice of 16 luminances, but mode 10 can support only eight — the even numbered luminances. If you select an odd-number, it will be rounded down to the next even number. To determine the luminance number, start with zero for the position of the arrow when the luminance screen first appears. This looks like it isn't pointing to a box, but like the other screens it is pointing to the selection that is being used to color the background of that section of the screen. Add one for each block to the right. The above example selected eight, but nine would give the same result because nine would round down to eight, which is the next lower even number.

One more choice that will affect screen colors is increment. You may have increments of 1 or 2. An increment of 1 is normal — the cursor moves one dot location at a time. An increment of 2 causes the cursor to move two dots at a time, putting a dot in every other dot location. This can be used to create shading effects (it was used to create the picture of the cylinder) or to combine up to four different colors in one area to get the effect of extra colors. Much experimenting is required to understand the possibilities that occur when using an increment of 2 in combination with the other features of this program.

Other Options

Press the OPTION key for the "Help" screen. This will give you a list of all the functions of this program. The ones on the left side have been explained. The ones on the right use the keyboard keys. R, L, and C help you create solid objects on the screen.

To see how the fill function works, first move the cursor to the left of the lines you have created on the screen. Any movement of the joystick will cause the "Help" screen to disappear and the drawing to return. Move the cursor to the left and up almost to the highest part of your drawing. Now press the R key. Then try drawing a line straight down by holding the trigger button with the joystick toward you. Continue until you have gone a line or two below the lines on the screen.

The "fill right" function fills in the space to the right of the cursor position with the selected color. The fill area stops when it reaches either a dot of the same selected color or, as you saw at the bottom, the edge of the screen. Try moving the cursor to the right of your drawing and use the L key for a left fill. The same thing should happen except that the fill lines go left instead of right.

Notice that the television sends out beeps while the right or left fill function is in effect. This is to warn you that if you hit the trigger your cursor is going to shoot out a line instead of a dot. To return to the single dot mode, press C. The beeping should stop and you will be back in the initial mode.

The last option left on the "Help" screen is the load/save option. The program is set up to load and save on disk, but it can be modified to use a cassette if that is what you have. It should work the same way except the file name request must be replaced by cassette commands. This modification is explained in Part II.

Pressing D will cause everything to stop on the screen for a few seconds, which means that the cursor will stop blinking. After that brief pause, the screen will be in the normal text mode and it will display a three-item selector entitled "Disk Transfers." The options are save, load, or return to the picture. Press 1 to save your picture on disk; press 2 to load one that you have previously saved. In either case you will be asked for the file name. This must be in the correct form for disk files — up to eight characters plus (optionally) a period followed by up to three characters. All characters must be upper-case letters or numbers.

When the load or save is complete, the disk transfers menu will reappear. If you loaded a picture, the new picture is now the "current" picture.

Save your pictures under different names because there is no checking in the program. If you name the picture with the same name as another file on the disk, the picture will replace the other file. When you do save a picture, a copy remains in memory as the "current" picture. The only way to change the "current" picture, other than drawing over it, is to load another picture from disk.

Mode 10 Drawings

The accompanying pictures show a few things that can be done with mode 10. Since there are nine selectable colors, as opposed to two or four

available in other map modes, shading can be used to effect depth. Mode 9 has more shading available (16 shades in all) but has only one color. Mode 11 has 16 colors, but they are all the same shade. Mode 10 allows you to select the hue and luminance of each of its nine colors.

The block drawing (see figure 1) shows how depth can be simulated by colors. The explanation that follows tells how to draw a block like the one pictured. The dimensions are approximate, so don't bother counting pixels!

To start the block, first push SYSTEM RESET and reRUN the program. That is the simplest way to clear the screen. Draw a line straight up as tall as you want the block. Next, draw one to the left about the same length. Now press R and draw a line down as far as the bottom of the first line. With the fill on, this should be easy to see. When done, press C to cancel the fill.

Move the cursor over to the lower right corner of the box plus one dot to the right. This will be the beginning of the drawing of the side of the box. Press START to choose another color. Move the pointer to the second color on the bar, which is a light pink, and press the trigger button.

The next part may be a little difficult for anyone not experienced in Atari games. The bottom of the side of this box is a diagonal line. Push the trigger down and position the joystick on the upper-right diagonal and draw a line as long as you want for the side of the box. If you've made a mistake in anything so far hit START, select the background color by pressing the trigger without moving the joystick, and draw over your mistake to correct it.

At the end of the diagonal, draw a line straight up. This line should be the same length as the sides of the front of the box, but just approximate that for now; corrections can be made later.

Bring the cursor to one pixel right of the upper-right corner of the front of the box to start the top edge of the side. Do not draw when you are moving the cursor to that point (don't hold down the trigger). Next, press R again to start the right fill function. Draw a fill line down to where you started drawing with this color — the lower-right corner of the front of the box. Actually, you should stop one pixel before that or you will get a pink line all the way to the edge of the screen. If that happens, leave it for corrections later.

Return to the upper end of the line from which you just filled (next to the upper-right corner of the front of the box). Leave the right fill on (do not press C yet). Draw a diagonal up and to the right and you will get a filled in triangle to form the top of that side of the box. Do not go beyond the rightmost part of your picture if your line was too short.

Cancel the fill now by pressing C. Move the cursor around using the trigger to place any dots you may have missed during that last fill. Next, press START and press the trigger to select the background color. Use the joystick and trigger to

(Continued on page 71)

Listing 1

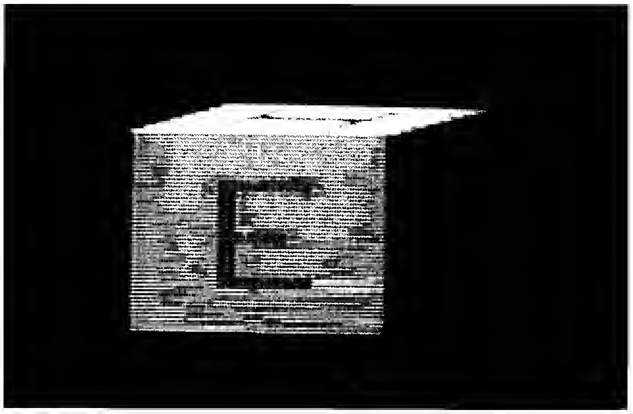
```
10 REM *** MODE 10 PAINTER ***
12 REM ***    PROGRAM      ***
14 REM ***                 ***
16 REM *** Designed by     ***
18 REM ***  Paul S Swanson ***
20 REM ***                 ***
22 ? "}<29><29><29>        MODE 10 PAINTER
 PROGRAM"
24 ? "          FOR ATARI COMPUTERS"
26 ? :? "Program by Paul S. Swanson"
28 ? :? :? "Initializing..."
30 REM +++ INITIALIZATION  +++
40 REM — JOYSTICK READ TABLE —
50 DIM JOY(15,1),A$(2)
60 FOR RDG=1 TO 15:FOR DIR=0 TO 1:READ JOY:JOY(RDG,DIR)=JOY:
 NEXT DIR:NEXT RDG
70 DATA 0,0,0,0,0,0,0,0,1,1,1,-1,1,0,0,0,-1,1,-1,-1,-1,
 0,0,0,0,1,0,-1,0,0
80 REM — POSITION STRINGS —
90 REM — ON 1K BOUNDARY   —
100 DIM X$(1):A=ADR(X$):B=INT(A/1024+1)*1024:DIM XX$(B-A-1):
 PMSTART=B/256
110 REM — DISPLAY LISTS   —
120 DIM HELPDL$(64),SELDL$(64)
130 REM — SCREEN AREAS    —
140 DIM HELPSC$(256),SELSC$(256)
150 REM — INITIALIZE DL'S —
160 HELPDL$="ppppppppB<0><0><2><2><2
 ><2><2>A"
170 SELDL$="pppppppp0<0><0><15><15><15
 ><15><15>pA"
180 ADRSETUP=5000
190 A=ADR(HELPDL$):GOSUB ADRSETUP:HELPDL$(LEN(HELPDL$)+1)=A$
200 A=ADR(SELDL$):GOSUB ADRSETUP:SELDL$(LEN(SELDL$)+1)=A$
210 REM — INITIALIZE SCREENS —
220 HELPSC$=" ":HELPSC$(256)=" ":HELPSC$(2)=HELPSC$
230 SELSC$="<0><0><0><0><0><0
 ><0><0>"
240 FOR I=17 TO 255 STEP 17
250 FOR J=1 TO 2:SELSC$(LEN(SELSC$)+1)=CHR$(I):NEXT J:NEXT I:
 SELSC$(39,40)="<0><0>"
260 SELSC$(256)=" ":SELSC$(41)=SELSC$
270 REM — PUT SCREEN ADDRESSES
          INTO DISPLAY LISTS
280 A=ADR(HELPSC$):GOSUB ADRSETUP:HELPDL$(10,11)=A$
290 A=ADR(SELSC$):GOSUB ADRSETUP:SELDL$(10,11)=A$
300 REM — INITIALIZE PLAYER2 —
310 DIM PL2$(128)
320 PL2$="<0>":PL2$(128)="<0>":PL2$(2)=PL2$
330 REM — HELP SCREEN TEXT —
340 HELPSC$(1,40)="<14><14><14><14><14
 ><14><14><14><14><14><14><14
 > HELP SCREEN <14><14><14><14><14
 ><14><14><14><14><14><14><14
 ><14><14><14>"
350 HELPSC$(41,80)="OPTION - HELP screen    |R Fill
 right   "
360 HELPSC$(81,120)="SELECT - Color Selection|L Fill
 left    "
370 HELPSC$(121,160)="START - Change Pen Color|C Cancel
 fill  "
380 HELPSC$(161,200)="1,2 - Increment         |D
 load/save    "
390 FOR I=1 TO LEN(HELPSC$):N=ASC(HELPSC$(I)):N1=(N>27):
 N=N-N1*128
392 N=(N-32)*(N>31 AND N<96)+(N+64)*(N<32)+N*
 (N>95)+N1*128:HELPSC$(I,I)=CHR$(N):NEXT I
400 REM —— SET UP MODE 10 SCREEN
410 GRAPHICS 10
420 REM —— USE RANDOM COLORS ——
430 COL=25:FOR REG=704 TO 712
440 POKE REG,COL:COL=COL+25:NEXT REG
450 POKE 704,0
460 REM — DEFINE CONSTANTS —
470 CONSOL=53279
480 CBASE=704
490 DMACTL=559
500 GRACTL=53277
510 HPOSP1=53249
520 PMBASE=54279
530 SIZEP1=53257
540 BEGIN=1000
550 KB=764
560 GTIA=623
570 NMIEN=54286
600 REM — INSTALL DLI ROUTINE —
```

(Continued on next page)

Listing 1 *Continued*

```
610 RESTORE 7000:LOC=1536
620 READ N:IF N<256 THEN POKE LOC,N:LOC=LOC+1:GOTO 620
630 POKE 512,0:POKE 513,6
640 REM — ALTERNATE SCREENS —
650 DIM ALTSC1$(256),8UFF$(8192)
660 ALTSC1$="<0>":ALTSC1$(256)="<0>":
 ALTSC1$(2)=ALTSC1$
670 FOR I=1 TO 240 STEP 40:ALTSC1$(I,I+23)=SELSC$(I,I+23):
 NEXT I
900 REM — OTHER DIMS —
910 DIM RCOL(9),F$(12),Q$(40),FILE$(14),LINE$(80)
920 REM — INITIALIZE COUNTERS, ETC.
930 X=39:Y=96
940 UNDERCURSOR=0
950 CURSORFLAG=0
960 CURSORCOUNT=0
970 SELCOLOR=1
980 FLASHCOUNT=0
982 INCREMENT=1
990 REM ***                    ***
992 REM *** MAIN PROGRAM TEXT ***
994 REM ***                    ***
996 REM — READ JOYSTICK/CONSOL —
1000 STK=STICK(0):CURSORCOUNT=CURSORCOUNT+1:IF CURSORCOUNT
 <4 THEN 1060
1002 IF FILLFLAG=0 THEN 1040
1010 FLASHCOUNT=6-FLASHCOUNT
1020 SOUND 0,60,10,FLASHCOUNT:FOR DELAY=1 TO 2:NEXT DELAY
1030 SOUND 0,0,0,0
1040 CURSORFLAG=1-CURSORFLAG:COL=UNDERCURSOR+CURSORFLAG:
 IF COL>8 THEN COL=0
1050 COLOR COL:PLOT X,Y:CURSORCOUNT=0
1060 IF STK<>15 OR STRIG(0)=0 THEN 1080
1070 SWITCH=PEEK(CONSOL):IF SWITCH<>7 THEN 4000
1072 IF PEEK(KB)<>255 THEN 3000
1074 GOTO BEGIN
1080 POKE 77,0
1110 COLOR UNDERCURSOR:IF STRIG(0)=0 THEN COLOR SELCOLOR
1120 PLOT X,Y
1130 REM —— MOVE CURSOR ROUTINE —
1140 X=X+JOY(STK,0)*INCREMENT:Y=Y+JOY(STK,1)*INCREMENT
1150 X=X-INT(X/80)*80:Y=Y-INT(Y/192)*192
1160 LOCATE X,Y,UNDERCURSOR
1170 CURSORFLAG=0:CURSORCOUNT=4:IF FILLFLAG=0 OR STRIG(0)=1
 THEN GOTO BEGIN
1172 REM — FILL ROUTINE —
1180 X1=X:COLOR SELCOLOR
1190 X1=X1+FILLFLAG*INCREMENT:IF X1>79 OR X1<0 THEN
 GOTO 8EGIN
1200 LOCATE X1,Y,TESTEND:IF TESTEND=SELCOLOR THEN GOTO 8EGIN
1210 PLOT X1,Y:GOTO 1190
2990 REM —
2992 REM — KEYBOARD INTERPRET ROUTINE
2994 REM —
3000 N=PEEK(KB):POKE KB,255:IF N=40 THEN FILLFLAG=1:
 GOTO BEGIN
3010 IF N=0 THEN FILLFLAG=-1:GOTO BEGIN
3012 IF N=31 OR N=30 THEN GOTO 8000
3020 IF N=18 THEN FILLFLAG=0:GOTO 8EGIN
3030 IF N<>58 THEN GOTO 8EGIN
3040 GOSUB 20000
3050 GRAPHICS 0:? "          DISK TRANSFERS":?
3060 ? " <1>  SAVE PICTURE ON DISK"
3070 ? " <2>  LOAD PICTURE FROM DISK"
3080 ? " <3>  RETURN TO CURRENT PICTURE":?
3090 ? "PRESS NUMBER OF SELECTION—";
3100 CLOSE #3:OPEN #3,4,0,"K:":GET #3,N:CLOSE #3
3110 N=N-48:IF N<1 OR N>3 THEN 3100
3120 GOTO N*100+3100
3200 ? "}         SAVE PICTURE ON DISK":? :DIRECTION=8:
 GOSUB 10000:? "SAVING PICTURE":TRAP 40000
3210 FOR I=0 TO 8:? #3;RCOL(I):NEXT I
3220 FOR I=1 TO 8160 STEP 80:? #3;BUFF$(I,I+79):NEXT I
3230 CLOSE #3:GOTO 3050
3300 ? "}        LOADING PICTURE FROM DISK":? :DIRECTION=4:
 ? :GOSUB 10000:? "LOADING PICTURE"
3310 FOR I=0 TO 8:INPUT #3,RCOL:RCOL(I)=RCOL:NEXT I
3320 FOR I=1 TO 8160 STEP 80:INPUT #3,LINE$:BUFF$(I,I+79)=
 LINE$:NEXT I
3330 CLOSE #3:GOTO 3050
3400 GRAPHICS 10:FOR I=0 TO 8:POKE I+C8ASE,RCOL(I):NEXT I
3410 FOR I=0 TO 8190 STEP 256:A=USR(ADR(Q$),8UFF+I,SCREEN+I):
 NEXT I
3420 LOCATE X,Y,UNDERCURSOR:GOTO 8EGIN
3989 GOTO 8EGIN
3990 REM —
3992 REM — FUNCTION KEY INTERPRETER
3994 REM —
4000 FOR I=1 TO 7:I=PEEK(CONSOL):NEXT I:GOSUB 5020:
 MODERES=PEEK(GTIA)
4010 ON SWITCH GOTO BEGIN,BEGIN,4100,BEGIN,4200,4300,BEGIN
4100 POKE GTIA,0:A=LEN(HELPDL$):POKE 560,ASC(HELPDL$(A-1)):
 POKE 561,ASC(HELPDL$(A))
4102 IF PEEK(CONSOL)<>7 THEN 4102
4110 IF STICK(0)=15 AND PEEK(KB)=255 AND PEEK(CONSOL)=
 7 THEN 4110
4120 POKE GTIA,MODERES:GOSUB 5030:GOTO BEGIN
4130 GOTO BEGIN
4140 IF STRIG(0)=1 THEN 4120
4150 GOTO 4140
4200 A=LEN(SELDL$):POKE 560,ASC(SELDL$(A-1)):POKE 561,
 ASC(SELDL$(A))
4210 A=ADR(ALTSC1$):GOSUB 5000:SELDL$(10,11)=A$
4220 MSG=6010:MAXSEL=8:GOSUB 5040:COLNO=SELECTION
4230 A=ADR(SELSC$):GOSUB 5000:SELDL$(10,11)=A$:COLSAV=
 PEEK(C8ASE+8):POKE CBASE+8,8
4240 POKE GTIA,192:MSG=6020:MAXSEL=15:GOSUB 5040
4250 POKE C8ASE+8,SELECTION*16:COLUSED=SELECTION
4260 POKE GTIA,64:MSG=6030:MAXSEL=15:GOSUB 5040
4270 POKE CBASE+8,COLSAV:COLUSED=COLUSED*16+SELECTION
4280 POKE CBASE+COLNO,COLUSED:GOTO 4140
4300 A=LEN(SELDL$):POKE 560,ASC(SELDL$(A-1)):POKE 561,
 ASC(SELDL$(A))
4310 A=ADR(ALTSC1$):GOSUB 5000:SELDL$(10,11)=A$
4320 MSG=6000:MAXSEL=8:GOSUB 5040
4330 A=ADR(SELSC$):GOSUB 5000:SELDL$(10,11)=A$
4340 SELCOLOR=SELECTION:GOTO 4140
4990 STOP
4992 REM +++
4994 REM ——  SUBROUTINES ——
4996 REM ——
4998 REM ——  Conv't A to address
             in A$
4999 STOP
5000 HI=INT(A/256):LO=A-HI*256
5010 A$=CHR$(LO):A$(2)=CHR$(HI):RETURN
5020 SHI=PEEK(561):SLO=PEEK(560):RETURN
5030 POKE 561,SHI:POKE 560,SLO:RETURN
5040 RESTORE MSG:READ F$
5050 PL2$(50,56)="<8><28>*I<8><8><8
 >":8ASE=58
5060 FOR I=1 TO LEN(F$):N=(ASC(F$(I))-32)*8+57344
5070 FOR J=0 TO 7:PL2$(J+BASE,J+8ASE)=CHR$(PEEK(J+N)):NEXT J
5080 BASE=8ASE+8:NEXT I
5090 POKE NMIEN,192:POKE DMACTL,42:POKE GRACTL,2:
 POKE PMBASE,PMSTART
5100 POKE SIZEP1,0:SELECTION=0:POKE 512,0:POKE 513,6:
 MAXSEL=MAXSEL+1
5110 POKE HPOSP1,SELECTION*8+72
5120 STK=STICK(0):IF STK=15 AND STRIG(0)=1 THEN 5120
5130 IF STRIG(0)=0 THEN POKE HPOSP1,0:PL2$(75)=PL2$(74):
 RETURN
5140 SELECTION=SELECTION+(STK=7)-(STK=11)
5150 SELECTION=SELECTION-INT(SELECTION/MAXSEL)*MAXSEL:
 POKE HPOSP1,SELECTION*8+72
5160 SOUND 0,135,10,6:FOR DELAY=1 TO 50:NEXT DELAY:
 SOUND 0,0,0,0
5170 GOTO 5120
6000 DATA COLR
6010 DATA CHGE
6020 DATA HUE
6030 DATA LUM
7000 DATA 72,169,14,141,19,208,169,0,141,10,212,141,27,208,
 141,26,208,104,64,256
8000 INCREMENT=32-N:GOTO 8EGIN
10000 ? "ENTER FILE NAME - MAX. 8 CHARACTERS:"
10010 INPUT F$
10020 IF LEN(F$)<1 THEN 11000
10030 TRAP 11000
10040 FILE$="D:":FILE$(3)=F$
10050 OPEN #3,DIRECTION,0,FILE$:RETURN
11000 ? "}ERROR - NOT A VALID NAME":FOR I=1 TO 300:NEXT I:
 CLOSE #3:GOTO 3050
20000 Q$="hh<133>Oh<133>Nh<133>Qh
 <33>P <0>H1N<145>P@<0>Pw'":
 BUFF$(B192)="<0>"
20010 DLIST=PEEK(560)+PEEK(561)*256:SCREEN=PEEK(DLIST+4)+
 PEEK(DLIST+5)*256:8UFF=ADR(8UFF$)
20020 FOR I=0 TO 8190 STEP 256:A=USR(ADR(Q$),SCREEN+I,
 BUFF+I):NEXT I
20030 FOR I=0 TO 8:RCOL(I)=PEEK(I+C8ASE):NEXT I
20040 RETURN
```

(Note: Underlined text indicates inverse video)



erase any mistakes and "clean up" your drawing.

The last part of the box is the top. Start at the pixel just above the upper-left corner of the front of the box. Use START to select the third color. This should be purple. Don't worry about shading and colors at this point as they will be changed after the box is drawn. Draw a diagonal up and to the right from this point. Extend it at least as far up as the diagonal at the other end of the top of the box. Move the cursor to the pixel just above the top of the leftmost edge of the side of the box. Press L to start the left fill and draw a diagonal line up and to the right as far as the rightmost edge of the side of the box. Use START to get the background color to correct any errors and to erase the overrun of the left edge of the top of the box.

Now is the time to choose the colors. As in the picture, the top will be the brightest, then the front, and the side will be the darkest. Also, you need to lighten the background and make one of the other colors black so that you can put a shadow on the screen.

Press SELECT and press the trigger to select the background first. When the 16 colors appear, press the trigger again to choose black. For the luminance, move the arrow over two boxes and press the trigger to select a slightly lighter shade of grey.

For the first color, which is the front of the box, select color 15 by moving the arrow one box to the left, which will cause it to "wrap" to the right edge. Select the eighth luminance box for this. The second color corresponds to the side of the box. Select the same hue with a luminance of four. For the third color, which is the top of the box, select the same color with a luminance of twelve. Lastly, select the fourth color, which hasn't yet been used, as black with a luminance of zero.

Use START to "load your brush" with this fourth color to plot out the shadow. Move the cursor down to just right of the lower-right corner of the front so that it is over the first pixel to the right of the corner that is in the background color. Draw a line from here to the right as long as you want the shadow. Be sure to leave some room to the right of this point for a diagonal line that will be drawn next.

From this point, draw a diagonal line up and to the right to parallel the bottom of the side of the box. Since you are estimating, it is better to make this line a few dots too long than too short. From there, draw a horizontal line to the left until you reach the box itself.

To fill the shadow, push R for right fill, then draw along the edge of the box down to where you started outlining the shadow. Press C to cancel the fill.

The basic box is now complete. Use the START function to select the colors needed to make any touch-up corrections to your box before you go on.

Dressing It Up

The box in the picture is dressed up with letters and an indication of shading toward the edges of the sides. Only five of the nine colors were used to build the basic box. Use these and the other four, selected any way you want, to put letters and shading on the box.

You can also outline a table on which the box can sit. First, outline the box and shadow with the color of the table, then draw the outline of the top of the table, using the fill commands in the proper places to extend the color of the table across the screen. Use a darker shade of that color for the sides and/or legs of the table.

Next Month

Now that you have used this painter program, you can probably think of some extra functions that you want to add to improve it. The program was designed to allow additions. Next month Part II explains how the program works using all three of the GTIA modes as well as custom display lists, screen images in strings, and a player. The description of the program shows where extra functions can be added and suggests a few. Play with the program this month so that you are familiar with its functions, and make sure you have this copy of MICRO handy when next month's arrives!

Bulletin Boards

MODE10 Atari Painting Program may be down loaded from one of the following bulletin boards:

- Atari Message and Information Service (AMIS), Cambridge, MA (617) 497-9054.
- Boston Bullet, Boston MA (617) 266-7789.
- AMIS North Shore, Lynn MA (617) 595-0211.

MICRO

Paul Swanson, our Atari columnist, has published articles in several microcomputer magazines and has authored a book on disk techniques. You may contact him at 97 Jackson St., Cambridge, MA 02140.

computer mail order west

computer mail order

PRINTERS

SMITH CORONA

Call on $50.00 Factory Rebate!

TP-1	$499.00
Tractor Feed	$129.00
Ultrasonic I Typewriter	$439.00

C. ITOH (TEC)

Gorilla	$209.00
Prowriter 8510P	$379.00
Prowriter 8510S	$579.00
Prowriter II-P	$699.00
Prowriter II-S	$749.00
Starwriter F10-40	$1259.00
Printmaster F10-55	$1649.00

OKIDATA

Call for price and information of OKIDATA 82A, 83A, 84(Parallel), 84(Serial), 92 & 93.

EPSON

Call for price and information on EPSON RX80, MX80, MX80FT, FX80 & MX100.

IDS

MicroPrism	$649.00
132 (Fully Configured)	$1599.00
80 (Fully Configured)	$1399.00

Call for other configurations.

STAR

Gemini 10	$339.00
Gemini 15	$489.00
Serial Board	$75.00

DAISYWRITER

2000 (Letter Quality)	$1149.00
Tractor Feed	$109.00

DIABLO

620	$999.00
630	$1769.00

TeleVideo

TERMINALS

910	$559.00
912C	$689.00
920C	$739.00
925C	$719.00
950	$929.00

COMPUTERS

800A	$1259.00
802	$2649.00
802H	$4695.00
806/20	$4949.00
816/40	$8999.00
803	$1949.00
1602G	$3399.00
1603	CALL

NEC

COMPUTERS

8000	CALL
8001A	$669.00
8031	$669.00
8012	$549.00
8801	CALL

PRINTERS

8023	$419.00
7710/7730	$2249.00
3510/3530	$1549.00

MONITORS

JB-1260	$119.00
JB-1201	$149.00
JC-1212	$299.00
JC-12-202	$299.00
JC-1203	$549.00

EAGLE

Call on Eagle 8 Bit & 16 Bit Computers & Software

MODEMS

HAYES

Smart	$219.00
Smart 1200 (1200 Baud)	$519.00
Chronograph	$199.00
Micromodem 100	$309.00
Micromodem II	$279.00
Micromodem II(with term)	$299.00
Smart Com II	$99.00
Smart 1200B	CALL

NOVATION

J-Cat	$119.00
Cat	$144.00
D-Cat	$159.00
103 Smart Cat	$189.00
Apple Cat II	$279.00
103/212 Smart Cat	$439.00
212 Apple Cat II	$609.00
Apple Cat II 212 Upgrade	$309.00

ANCHOR

Mark I (RS-232)	$79.00
Mark II (Atari)	$79.00
Mark III (T.I.-99)	$109.00
Mark IV (CBM-PET)	$125.00
Mark V (Osborne)	$95.00
Mark VI (IBM-PC)	$179.00
Mark VII (Auto Ans./Auto Dial)	$99.00
TRS-80 Color Computer	$99.00
9 Volt Power Supply	$9.00
Mark VIII	CALL

MONITORS

AMDEK

300G	$159.00
300A	$169.00
310A	$169.00
Color I	$279.00
Color I plus	299.00
Color II	$439.00
Color III	$349.00

BMC

12AU 12" Green	$79.99
1401 13" Color (Mid-Res)	$369.00
9191 U 13" Composite	$329.00

TAXAN

RGB 1 (Hi-Res)	$299.00
RGB III	$499.00
12 N Green	$119.00
12 A Amber	$125.00

ZENITH

ZVM 121	$95.00
ZT-1 Terminal	$369.00

U.S.I.

Pi 1, 9" Green	$99.00
Pi 2, 12" Green	$119.00
Pi 3, 12" Amber	$159.00
Pi 4, 9" Amber	$139.00
1400C 14" Color	$299.00

PANASONIC

JR200U32K Pers. Computer ... $309.00

MONITORS

TR-120 12" Hi-Res Green	$159.00
CT-160 10" Duet Mode Color	$299.00
DT-D1000 10" RGB	$349.00
DT-D1300 13" RGB/Compos	$429.00

IBM

NEC

3550 PRINTER...$1949

PERCOM/TANDON

DRIVES

5¼" 160K Disk Drive	$249.00
5¼" 320K Disk Drive	$299.00

AMDEK

310A Amber Monitor	$169.00
Amdisk III (3" Drive)	$499.00
DXY Plotter	$599.00
Color II	$439.00

AST

Mega Plus 64	$399.00

QUBIE

PC Keyboard	$219.00

SOFTWARE

MicroPro WordStar/MailMerge	$319.00
I.U.S. Easywriter II	$249.00
I.U.S. Easyspeller	$129.00
Peach Package (GL/AP/AR)	$419.00

PROFESSIONAL SOFTWARE

PC Plus (Word Processing)	$319.00

CONTINENTAL SOFTWARE

1st Class Mail/Form Letter	$89.00
The Home Accountant Plus	$109.00

SYNAPSE

File Manager	$119.00

PFS	APPLE	IBM
File	$89.00	$99.00
Report	$89.00	$89.00
Graph	$89.00	$99.00
Write	n/a	$99.00

READY FORMS

1" or 2" Address Labls (Tract. Feed)	$9.95
15" Report Paper (Tract. Feed)	$24.95
8½" Blnk Wht Paper (Tract. Feed)	$19.95
8½" Blnk Env (Tract. Feed)	$14.95

SANYO

MB1000 Computer ... $1599.00

INCLUDES FREE SOFTWARE! MicroPro, WordStar, Calcstar, Mail Merge & Report Star

MB 160 Add on Drive	$539.00
5500 Letter Quality Printer	$699.00

TIMEX SINCLAIR 1000

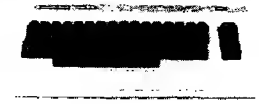
$39.95

16K Memory Module	$44.95
Vu-Calc	$17.95
Check Book Manager	$13.95
The Organizer	$14.95
The Budgeter	$13.95
Stock Option	$14.95
Loan & Mortgage Amortizer	$12.95
Mindware Printer	$109.00

ORSYTE SOFTWARE

Graphics	$12.99
Home Budgeter	$12.99
Home Inventory	$12.99
Income Tax	$14.99
Mega Mind	$15.99
Selvo	$12.99
The Quiz	$12.99
Words	$12.99

commodore



1520 Color Printer/Plotter	$169.00
1525 80 Column Printer	$339.00
1530 Datasette	$69.00
1541 Single Disk Drive	$339.00
1600 VIC Modem	$95.00
1610 VIC Term 40	$49.00
1650 AD/AA Modem	$159.00
1701 14" Color Monitor	$269.00
1311 Joysticks (each)	$5.99
1312 Paddles	$11.99
1110 VIC 8K	$42.00
1111 VIC 16K	$69.00
1011 RS232 Interface	$42.00
1211 Super Expander	$53.00
1906 Super Alien	$23.00
1910 Radar Rat Race	$23.00
1917 VooDoo Castle	$29.00
1922 Cosmic Cruncher	$35.00
1923 Gorf	$29.00
1924 Omega Race	$30.00
110 VIC 20 Reference Guide	$15.00
CBM 64 Reference Guide	$18.00

EASY BUSINESS SERIES 64

Easy File	$79.00
Easy Finance	$39.00
Easy Mail	$39.00
Easy Script	$79.00
Word Machine/Name Machine	$23.00

PROGRAMMER SERIES 64

Assembler	$39.00
Logo	$79.00
Pilot	$79.00
Pet Emulator	$25.00
Screen Editor	$25.00
Video/Music Support	$39.00

ART AND MUSIC SERIES 64

Music Machine	$25.00
Music Composer	$25.00
Meta Music I	$79.00

COMMERCIAL DATA

Motor Mouse/20	$23.00
Centipede/20	$23.00
Froggee (20/64)	$23.00

CREATIVE SOFTWARE

Astro Blitz	$32.00
Black Hole	$32.00
Trashman	$32.00
Home Finance	$27.00
Home Inventory	$13.00

UMI

Amok	$30.00
Meteor Run	$40.00
Alien Blitz	$30.00
Viterm A	$16.00
The Alien	$16.00

CBM 64 $369

VIC 20 $99.

8032...... $599.00

64K Upgrade Kit	$269.00
8096	$849.00
9000	$999.00
B-128	$769.00
B-256	$969.00
4032	$599.00
4064	$569.00
2031	$299.00
8050	$949.00
8250	$1199.00
9060 (5 Meg. HD)	$1999.00
9090 (7.5 Meg HD)	$2199.00
4023	$379.00
8023	$569.00
6400 Let Qual. Prnt.	$1399.00
Spell Master	$149.00
ZRam adds CP/M & 64K	$549.00
Silicon Office	$749.00
Calc Result	$159.00
The Manager	$209.00
The Soft Rom	$129.00
Jinsam 800C/64	CALL
ADA1600 CBM to Parl Int.	$89.00
ADA1450 CBM to Ser Int.	$99.00

PROFESSIONAL SOFTWARE

Power	$79.00
Info Pro	$199.00
Word Pro 2 Plus	$159.00
Word Pro 3 Plus	$199.00
Word Pro 4 Plus	$299.00
Word Pro 5 Plus	$299.00
Administrator	$379.00
Word Pro 64	$69.95
Spell Checker	CALL

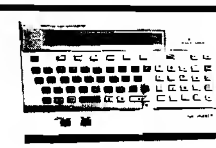
CARDCO VIC 20/CBM 64

Light Pen	$32.00
Cassette Interface	$29.00
Parallel Printer Interface	$64.00
3Slot Expans. Interface (20 only)	$32.00
6Slot Expans Interface (20 only)	$79.00

HES

Shamus (ROM)	$29.00
Protector (ROM)	$32.00
Robot Panic (ROM)	$29.00
Pirates (ROM)	$29.00
HES Writer (ROM)	$29.00

Call on our Large Selection of VIC 20 & C-64 Software, such as: EPYX, Microspec, Quick Brown Fox and Kansas City Software.



PC-1500 POCKET COMPUTER $169.

PC1250.... $89.00

SHARP

CE150 Printer, Plotter and Cassette Interface Unit	$172.00
CE152 Cassette Recorder	$62.00
CE155 8K Ram Expans. Mod.	$94.00
CE125 Printer/Micro Cass	$129.00
Statistics Pack	$49.00

computer mail order east

VISA

800-233-8950

In PA call (717)327-9575, Dept.715, 477 E. 3rd St., Williamsport, PA 17701

INTERNATIONAL ORDERS: All shipments outside the Continental United States must be pre-paid by certified check only. Include 3% (minimum $3.00) shipping and handling. **EDUCATIONAL DISCOUNTS:** Additional discounts are available from both Computer Mail Order locations to qualified Educational Institutions. **APO & FPO:** Add minimum $5.00 shipping and handling.

computer mail order

FRANKLIN

Call for price and information on the NEW FRANKLIN Computers!
Disk Drives, Software and System Specials ALSO AVAILABLE!

MICRO-SCI
DISK DRIVES FOR APPLE & FRANKIN

A2	$269.00
A40	$349.00
A70	$459.00
C2 Controller	$79.00
C47 Controller	$89.00

RANA DISK DRIVES

Elite 1 plus (Apple/Franklin)	CALL
1000 (Atari)	CALL

AMDEK DISK DRIVES

Amdisk I, 3" Mini Disk for Apple II & IIe	$229.00

SYSCOM
48K Color Computer
APPLE COMPATIBLE
$599.

VISICORP
for Apple, IBM & Franklin

Visidex	$189.00
Visifile	$189.00
Visiplot	$159.00
Visiterm	$89.00
Visitrend/Plot	$229.00
VisiSchedule	$229.00
Desktop Plan	$189.00
Visicalc(Apple II, CBM, IBM)	$179.00

Visicorp prices for IBM may vary slightly.

BRODERBUND

Apple Panic	$23.00
David's Magic	$27.00
Star Blazer	$25.00
Arcade Machine	$34.00
Choplifter	$27.00
Serpentine	$27.00

INFOCOM

Deadline(AP., IBM, AT.&C-64)	$35.00
Star Cross	$29.00
Zork I, II or III	$29.00

MPC

Bubdisk (128K Non Volitare)	$649.00

AXLON

Apple/Franklin 128K Ram	$299.00
Apple/Franklin Ram Disk	$729.00

KRAFT

Apple Joystick	$44.00

CONTINENTAL SOFTWARE

Tax Advantage (Apple/Atari)	$45.00
Home Accnt. (Apple/Atari)	$59.00
1st Cl. Mail/Form Letter(Apple)	$79.00
The Book of Apple	$14.95
The Book of Atari	$14.95
The Book of Apple Graphics	$14.95

PERCOM

DISK DRIVES FOR ATARI

AT88-S1	$399.00
AT88-A1	$299.00
AT88-S2	$649.00
RFD 40-S1	$549.00
RFD 40-A1	$349.00
RFD 40-S2	$889.00
RFD 44-S1	$679.00
RFD 44-S2	$999.00

FLOPPY DISKS

MAXELL

MD I (Box of 10)	$32.00
MD II (Box of 10)	$44.00
FD I (8")	$40.00
FD II (8" DD)	$50.00

VERBATUM

5¼" SS DD	$26.00
5¼" DS DD	$36.00

ELEPHANT

5¼" SS SD	$18.99
5¼" SS DD	$24.99
5¼" DS DD	$29.99

HEAD

Disk Head Cleaner	$14.95

SIRIUS

Bandits (Apple)	$28.00
Beer Run (Apple)	$24.00
Free Fall (Apple)	$24.00
Sneakers (Apple)	$24.00
Snake Byte (Apple)	$24.00
Fast Eddie (Atari)	$21.00
Turmoil (Atari)	$21.00
Deadly Duck (VIC)	$21.00

INTERFACES & ACCESSORIES

80 Column Apple Card	$159.00
Apple Paral. Printer Interface	$89.00
Atari 850 to Paral. Printer Cable	$29.00
RS232-RS232 Cables	$29.00
Centronics Paral.-Paral. Cables	$29.00

Call on IBM, Osborne, Apple and Franklin Interface Cards, Cables and Accessories.

HEWLETT PACKARD

41CV **$205.**

HP75...$749.

HP 41C	$148.00
HP 10C	$52.00
HP 11C	$89.00
HP 12C	$90.00
HP 15C	$90.00
HP 16C	$90.00

For HP41/41CV

HPIL Module	$99.00
HPIL Cassette or Printer	$359.00
Card Reader	$144.00
Extended Functions Module	$64.00
Time Module	$84.00
Mathmetics Pac	$28.50
Real Estate Pac	$40.00

HOME COMPUTERS

ATARI

400-16K..........$99.
Our Price $149.00, Get $50.00 rebate direct from Atari!

600XL-16K.....$149.

800-48K.........$389.
Our Price $489.00, Get $100.00 rebate direct from Atari!

1200-64K.......$449.
Our Price $549.00, Get $100.00 rebate from Atari!

ATARI 400 VERSIONS
32K...$174.*
48K...$199.*
64K...$259.*
*Non-Atari Ram

810 Disk Drive	$419.00
Inhome Keyboard/Atari 400	$39.00
1010 Program Recorder	$74.00
1020 40 Column Print/Plot	$269.00
1025 80 Column Printer	$469.00
1027 Letter Quality Printer	$299.00
1050 Double Density Drive	$379.00
830 Acoustic Modem	$159.00
850 Interface Module	$169.00
CX40 Pair Joysticks	$18.00
CX418 Home Manager Kit	$69.00
CX419 Bookkeeper Kit	$195.00
CX482 Educator Kit	$129.00
CX483 Programmer Kit	$54.00
CX488 Communicator Kit	$229.00
CX7101 Entertainer Kit	$69.00
Invitation to Programming I	$18.00
Invitation to Programming II	$20.00
Invitation to Programming III	$20.00
4002 Basic Language	$42.00
4003 Assembler Editor	$47.00
8121 Micro Assembler	$89.00
8126 Micro Soft	$89.00
405 Pilot (Edu.)	$105.00
4018 Pilot (Home)	$72.00
8036 Atari Writer	$79.00
404 Word Processor	$119.00
5059 Visicalc	$159.00

ATARI

Pacman	$33.00
Centipede	$33.00
Defender	$33.00
Galaxian	$33.00
Missile Command	$29.00
Star Raiders	$33.00
Caverns of Mars	$32.00
Dig Dug	$33.00
Donkey Kong	CALL
E.T. Phone Home	$39.00
Eastern Front (1941)	$39.00
Qix	$33.00
Superman III	$39.00
Star Trux	$33.00
Asteroids	$29.00
Basketball	$29.00
Computer Chess	$29.00
Juggles House	$23.00
My First Alphabet	$29.00

APX

Text Formatter	$18.50
Family Budgeter	$16.50
Eastern Front	$24.00
Family Cash	$18.50
Jukebox	$13.50
Downhill	$18.50
Outlaw	$18.50
Holy Grail	$24.00
Player Piano	$18.50
Keyboard Organ	$18.50
Number Blast	$13.50
Frogmaster	$18.50
747 Land Simulator	$18.50
Bumper Pool	$13.50

ON-LINE

Jawbreaker	$27.00
Softporn	$27.00
Wizard and Princess	$29.00
The Next Step	$34.00
Mission Asteroid	$22.00
Mousekattack	$31.00
Frogger	$31.00
Cross Fire (ROM)	$36.00

Some ON-LINE Products are also available for Apple & IBM. Call for information!

SYNAPSE

File Manager 800 plus	$69.00
Chicken (ROM)	$34.00
Picnic Paranoia (ROM)	$34.00
Claim Jumper (ROM)	$34.00
Slime (ROM)	$34.00
Shamus (ROM)	$34.00
Protector (ROM)	$34.00
Dodge Racer (C/D)	$26.00
Nautilus (C/D)	$28.00
Shadow World (C/D)	$28.00
Survivor (C/D)	$26.00
Oralbs (C/D)	$26.00
Necromancer (C/D)	$26.00
Pharoh's Curse (C/D)	$28.00
Fort Apocalypse (C/D)	$26.00
Assembler	$30.00

DATASOFT

Pacific Coast Highway	$25.00
Canyon Climber	$25.00
Tumble Bugs	$19.00
Shooting Arcade	$19.00
Clowns and Balloons	$25.00
Graphic Master	$30.00
Graphic Generator	$13.00
Micro Painter	$25.00
Text Wizard	$59.00
Spell Wizard	$39.00
Bishop's Square	$25.00
Sands of Egypt	$25.00
Moon Shuttle	$25.00
Zaxxon	$29.00
Basic Compiler	$59.00

ALIEN

Atari Voice Box	$119.00
Apple Voice Box	$149.00

MEMORY

Axlon 32K Ram	$65.00
Axlon 48K Ram	$109.00
Axlon 128K Ram	$299.00
Intec 32K Board	$74.00
Intec 48K Board	$99.00
Intec 64K Board (400 only)	$149.00

JOYSTICKS

Wico Joystick	$24.95
Famous Red Ball	$26.95
Apple Trackball	$59.00
Atari/VIC Trackball	$55.00
Apple Adaptor	$18.00

CBS

K-razy Shoot Out	$29.00
K-razy Kritters	$29.00
K-razy Antics	$29.00
K-star Patrol	$29.00
Stick Stand	$3.99

EPYX

Crush, Crumble & Chomp	$24.00
Crypt of the Undead	$24.00
Curse of Ra	$16.00
Datestones & Ryn	$16.00
Invasion Orion	$19.00
King Arthur's Heir	$24.00
Morloc's Tower	$16.00
Rescue at Rigel	$24.00
Ricochet	$16.00
Star Warrior	$29.00
Temple of Apshai	$29.00
Upper Reaches of Apshai	$16.00

SPINNAKER

Snooper Troops # 1	$34.00
Snooper Troops # 2	$34.00
Face Maker	$24.00
Story Machine	$24.00
Delta Drawing	$45.00
Rhymes and Riddles	$21.00
Kindercomp	$21.00

ROKLAN

Wizard of War (ROM)	$34.00
Deluxe Invader (ROM)	$29.00
Gorf (ROM)	$34.00

FIRST STAR

Astro Chase	$25.00

BIG 5

Miner 49er	$35.00

GAMESTAR

Baja Buggies	$24.95
Football	$24.95

BUDGECO

Raster Blaster	$24.95

L.J.K.

Letter Perfect 40/80 Col. Disk	$109.00
Letter Perfect 40 Col. ROM	$179.00
Letter Perfect 80 Col. ROM	$179.00
Data Perfect 40/80 Col. Disk	$99.00
Mail Merge	$21.95

CALL FOR APPLE/LJK PRODUCTS

BIT 3

80 Col. Card for Atari 800	$249.00

INTEC

Real Time Clock	$39.00

We also stock software by: Adventure International, Eduware, Creative Software, Dorsett Educational, IDSI, Romox and Artworx. Call for details and prices

computer mail order east

computer mail order west

Circle No. 42

800-648-3311

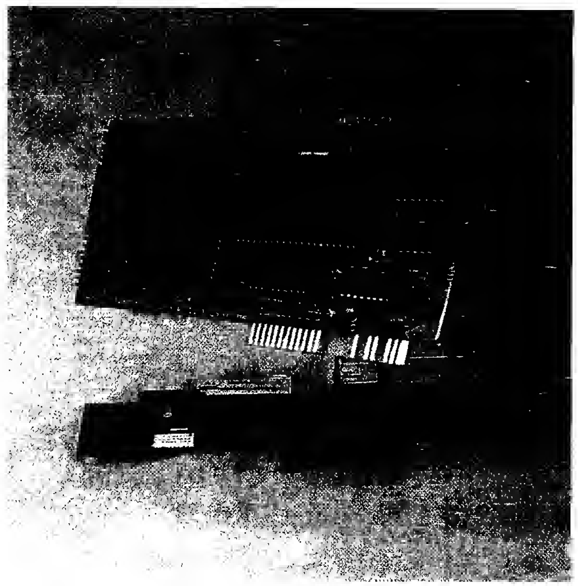
VISA

In NV call (702)588-5654, Dept. 715, P.O. Box 6689, Stateline, NV 89449

No risk, no deposit on C.O.D. orders. Pre-paid orders receive free shipping within the UPS Continental United States with no waiting period for certified checks or money orders. Add 3% (minimum $3.00) shipping and handling on all C.O.D. and credit card orders. Larger shipments may require additional charges. NV and PA residents add sales tax. All items subject to availability and price change. NOTE: We stock manufacturer's and third party software for most all computers on the market. Call today for our new catalog.

MEGAFLEX® ABILITY

You Pick The Disk System, MegaFlex Controls It!



WITH SOFTDRIVERS FOR A FLEXIBLE FUTURE!

MEGAFLEX—a universal floppy disk controller and modern alternative to the Apple drive system offering increased storage, improved reliability and . . . FLEXIBILITY.

Enjoy megabytes of online storage with your choice of micro, mini, or maxi drives—or even 6Mb with the Amlyn cartridge pack! Ideal for high-capacity storage now, winchester-disk backup later.

The MEGAFLEX secret is to autoboot softdrivers that match the needs of your drive system. All hardware functions are software-controlled. MEGAFLEX can match new drive capabilities without hardware changes. Drive-dependent ROMs have been eliminated.

APPLE *///*? OF COURSE!!

MEGAFLEX is compatible with BASIC, CP/M, Pascal, VISICALC, SOS and DOS-emulation on the Apple III, Apple II, Franklin Ace and Basis. All language features and operating system commands (LOAD, BRUN, etc.) are standard. If you can operate Apple drives you can operate MEGAFLEX! Your Apple software will run without modification too.

BRIDGE THE APPLE FORMAT BARRIER!

The MEGAFLEX diskette does what Apple's cannot—read and write diskettes from other computers! Software-controlled industry-standard IBM 3740 or System 34 type formats allow the MEGAFLEX library of reformatting software to read and write Altos, Radio Shack, Osborne, and IBM PC diskettes. (Call for the latest software details.)

MORE STORAGE, MORE UNIVERSAL FEATURES, LOWEST COST

MEGAFLEX with 8″ maxi or high density 5.25″ minis gives you 1.2 Megabyte of formatted data per diskette for 8 times the file and data size!

MEGAFLEX offers flexible software choices:

- data rate (250/500 Kbits per second),
- single and double density recording, and
- single/double sided drive operation (max 4 drives).

MEGAFLEX has the lowest chip count of any controller today! This means less power, a cooler Apple and better reliability.

Lowest price, highest performance, that's MEGAFLEX!

MEGABYTER™
A Division of SVA

11722 SORRENTO VALLEY ROAD
SAN DIEGO, CA
(619) 452-0101
TWX 910-335-2047 APPLE TWO SDG

TRADEMARKS CP/M—Digital Research

Circle No. 43

WORD DETECTIVE

by Bob Tripp

Word Detective is a game that challenges you to figure out why the computer accepts some words and not others. By typing in different words, you attempt to figure out what rule the computer is using to decide whether or not it likes a word. The game introduces the concept of the computer handling letters and words, as well as numbers. Detailed listings and instructions show you how to modify the program to use rules that you create.

Programming Concepts

While we normally think of computers as devices for dealing with numbers (referring to them sometimes as 'number crunchers'), they also can deal effectively with alphabetic material. One of the major uses of microcomputers today is as a word processor — a machine that can replace the typewriter and offer many kinds of support to the writer. Word processors can provide editing features such as character insert, character delete, line insert, line delete; they can move blocks of data, and more. Spelling-checking programs can eliminate most spelling errors. Text can be stored on cassette or disk, sent from one computer to another, and printed out in various formats. BASIC has a number of commands to assist in dealing with non-numeric information, and the most important ones are demonstrated in WORD DETECTIVE.

Processing ALPHABETIC Data

A CHARACTER is a single unit of information such as a letter (A d X), or a digit (0 1 2), a punctuation mark (! ? '), a mathematical symbol (= > < +), a special symbol ($ # [@), or various graphic symbols and control characters. BASIC can handle characters in a number of ways:

1. It can reference them directly, as in a PRINT string, with a string of characters enclosed in quotation marks: PRINT "THESE ARE CHARACTERS".
2. It can refer to them as a variable: C$ = "LETTER".
3. It can place them in arrays: A$(3) = "7", etc.

The STRING Concept

The VIC can work with a single letter, a word, a sentence, or a large portion of text as a single entity. This collection of letters (which can include digits and most special characters) is called a 'STRING'. BASIC keeps track of strings separately from other types of information, such as floating point and integer numbers. It names them with a one- or two-character name, like the numbers, but adds the dollar sign '$' to show that a string of characters is being referenced. Examples of string names include A$, Z4$, and BT$. A string may have *no* characters, in which case it is called a NULL string. A string may have a single character or may be very long, often limited only by the amount of memory the system contains.

In WORD DETECTIVE we demonstrate several different ways to handle strings. An examination of some of these methods should provide a basic understanding of the string processing capabilities of BASIC.

```
120 INPUT Z$ : IF LEN(Z$)=0 THEN 120
```

The INPUT statement accepts a string of characters from the keyboard until the RETURN key is pressed. It stores the entire string, which may be no characters (just a RETURN) or a NULL string, a single character, or many characters. In line 120, the string that has been typed is given the name Z$ so that it may be referenced by other parts of the program.

The LEN function determines the LENgth of the string that was INPUT into Z$. In line 120 it is used with an IF statement to check for a NULL or empty string. If the length of Z$ = 0, then *no* word was typed and the program tries again to get a word. If any characters have been typed, then the length of Z$ would not be zero.

```
130 IF Z$ = "THE RULE IS" THEN GOSUB 3000
    :X=R:GOTO 110
```

The entire contents of the Z$ string may be tested by comparing it to a string that is specified by a pair of quotation marks. If the Z$ string matches the string "THE RULE IS" exactly, character by character, then the subroutine at 3000 will be executed. If there is not an exact match, then the next instruction will be executed.

In addition to dealing with the entire text string, as in the INPUT, IF, or PRINT statements, BASIC can deal with portions of the total string. To work with part of the string, BASIC needs functions to isolate that part of the string that is of interest. Three functions are provided for this purpose:

1. LEFT$(Z$,N), which will isolate the LEFT N characters of the Z$ string;

Word Detective Listing

```
0 REM  WORD DETECTIVE
10 GOSUB 9000 : REM INIT
100 REM  MAIN PROGRAM
110 R=INT(RND(1)*5)+1:IF R=X THEN 110
120 INPUT Z$ : IF LEN(Z$)=0 THEN 120
130 IF Z$ = "THE RULE IS" THEN GOSUB 3000:X=R:GOTO 110
140 IF Z$ = "THE RULES ARE" THEN 4000
199 REM  PROCESS NORMAL WORD
200 J=0:D=0:V=0:C=0
210 F$ = LEFT$(Z$,1)
220 L$ = RIGHT$(Z$,1)
230 L = LEN(Z$)
240 IF INT(L) AND 1 THEN J=1
250 IF L = 1 THEN 290
260 FOR K = 1 TO L
270 IF MID$(Z$,K,1) = MID$(Z$,K+1,1) THEN D = 1
280 NEXT
290 FOR K = 1 TO L
300 V$ = MID$(Z$,K,1)
310 IF V$="A" OR V$="E" OR V$="I" THEN 340
320 IF V$="O" OR V$="U" OR V$="Y" THEN 340
330 C = C+1: GOTO 350
340 V = V+1
350 NEXT
999 REM  SELECT TEST
1000 ON R GOSUB 1100,1200,1300,1400,1500:GOTO 120
1100 IF J=0 THEN 2000
1110 GOTO 2100
1200 IF L=<5 THEN 2100
1210 GOTO 2000
1300 IF C>V THEN 2000
1310 GOTO 2100
1400 IF F$<L$ THEN 2100
1410 GOTO 2000
1500 IF D=1 THEN 2100
1510 GOTO 2000
1999 REM  CHANGE WORD COLOR
2000 PRINT "[CU]";"  ";"[GRN]";: REM  GREEN FOR HATE
2010 PRINT Z$;"[BLU]"
2020 RETURN
2100 PRINT "[CU]";"  ";"[RED]";:REM  RED FOR LOVE
2110 GOTO 2010
3000 REM  THE RULE IS
3010 PRINT "[CD][BLK]RULE #";R:PRINT: GOSUB 3100
3020 PRINT:PRINT "PRESS RETURN"
3030 PRINT "  TO CONTINUE: ";
3040 INPUT Z$:RETURN
3100 PRINT "[BLU]I HATE WORDS WITH[GRN] ":PRINT Y$(R-1):PRINT
3110 PRINT "[BLU]I LOVE WORDS WITH[RED] ":PRINT X$(R-1);
3120 PRINT "[BLU]": PRINT
3130 RETURN
4000 REM  THE RULES ARE
4010 PRINT "[CLR][BLK]WORD DETECTIVE RULES:[BLU]":PRINT
4020 FOR R = 1 TO 5:PRINT "[BLK]RULE #";R
4030 GOSUB 3110
4040 NEXT
4050 GOTO 110
9000 REM INITIALIZATION
9010 DIM X$(4),Y$(4)
9020 RESTORE
9030 FOR I = 0 TO 4
9040 READ X$(I),Y$(I): NEXT
9050 PRINT "[CLR][BLK]WORD DETECTIVE":PRINT "[BLU]"
9060 PRINT "IF I LIKE A WORD"
9070 PRINT "  I MAKE IT[RED] RED[BLU]":PRINT
9080 PRINT "IF I HATE A WORD"
9090 PRINT "  I MAKE IT[GRN] GREEN[BLU]":PRINT
9100 PRINT "FIGURE OUT MY RULE"
9110 PRINT "  BY TRYING DIFFERENT"
9120 PRINT "  WORDS.":PRINT
9130 PRINT "WHEN YOU HAVE FIGURED"
9140 PRINT "  IT OUT, OR YOU GIVE"
9150 PRINT "  UP, TYPE:""
9160 PRINT "[BLK]    THE RULE IS[BLU]"
9170 PRINT "  TO DISPLAY THE RULE."
9180 PRINT:PRINT "TO DISPLAY ALL OF THE"
9190 PRINT "  RULES, TYPE:"
9200 PRINT "[BLK]    THE RULES ARE[BLU]"
9210 R = RND(-TI)
9220 RETURN
10000 DATA "ODD NO. OF LETTERS","EVEN NO. OF LETTERS"
10010 DATA "5 LETTERS OR FEWER","6 LETTERS OR MORE"
10020 DATA "VOWELS => CONSONANTS","VOWELS < CONSONANTS"
10030 DATA "FIRST LETTER < LAST","FIRST LETTER => LAST"
10040 DATA "DOUBLE LETTERS","NO DOUBLE LETTERS"
READY.
```

Do not type those characters included in square brackets. Use these cursor or control keys:

[CU] SHIFT and CRSR up/down
[CD] CRSR up/down
[CLR] SHIFT and CLR/HOME
[BLK] CTRL and
[BLU] appropriate
[RED] color key
[GRN]

2. RIGHT$(Z$,N), which will isolate the RIGHT N characters of the Z$ string;
3. MID$(Z$,M,N), which will isolate the MIDdle N characters of the Z$ string starting at the character in position M.

Let us assume that the INPUT to the Z$ string was the word "SAMPLE".

```
210 F$ = LEFT$(Z$,1)
```

F$ is set equal to "S", the first character in "SAMPLE". The LEFT$ function (Z$,1) says to take one character from the left end of the string Z$.

```
220 L$ = RIGHT$(Z$,1)
```

L$ is set equal to "E", the last character in "SAMPLE". The RIGHT$ function (Z$,1) says to take one character from the right end of the string Z$.

```
300 V$ = MID$(Z$,K,1)
```

V$ is set equal to the character at position K in the string Z$: 'S' when K is 1, 'A' when K is 2, and so on up to 'E' when K is 6. The MID$(Z$,K,1) says to take one character from the K position in the string Z$.

In the above examples, a single letter has been isolated from the entire string. Substrings (parts of strings) may be isolated in the same manner. The number 1 would be changed to the length of the substring that was to be isolated — 2 for two characters, 3 for three characters, and so forth up to the length of the string.

Strings may be handled in arrays, just like numeric data. The array name must have a $ to indicate that the data is alphanumeric.

```
9040 READ X$(I),Y$(I): NEXT
```

This will READ the strings of DATA starting at line 10000 into two arrays, X$(I) and Y$(I).

```
10000 DATA "ODD NO. OF LETTERS",
      "EVEN NO. OF LETTERS"
```

X$(0) will be set to equal "ODD NO. OF LETTERS" and Y$(0) will be set equal to "EVEN NO. OF LETTERS". These individual strings. which are the written rules for the game, are then output by referencing them in a PRINT statement.

```
3100 PRINT "I HATE WORDS
     WITH" :PRINT Y$(R – 1)
     :PRINT
```

After printing the 'canned' message "I HATE WORDS WITH", the PRINT Y$(R – 1) statement will output the string referenced by R – 1. For example, if R = 1, then R – 1 will equal 0, and Y$(0) contains the string "EVEN NO. OF LETTERS" as shown above, and this is what will be printed.

Program Description

At line 10, the program goes to subroutine 9000 to initialize. The initialization procedure consists of the following steps:

Line 9010 DIMensions two string arrays, X$ and Y$.

Line 9020 RESTOREs the DATA pointer for the following READ statements so that the READ will start at the first DATA statement.

Lines 9030 to 9040 use a FOR...NEXT loop to READ the DATA statements into the string arrays dimensioned above.

Lines 9050 to 9200 are simply a series of PRINT statements that display the operating instructions for the program.

Line 9210 initializes the Random Number Generator (RND) by using the current time in the VIC built-in timer (TI) as the seed value.

Line 9220 RETURNs the program to the mainline code.

The mainline of the program, lines 100 to 1000, uses the random number generator to select the current 'RULE', accept the word from the user, analyze the word for a number of characteristics, and then evaluate the word relative to the rule.

Line 110 generates a random number and then makes sure it is not the same number as the previous number. If the new RULE number is the same as the old one, then another number is generated until a new RULE is selected.

Line 120 accepts INPUT from the user into the string variable Z$. If the user hits a RETURN with no word then the LENgth of Z$ will be = 0 and the program will restart by displaying the main instructions again.

Line 130: If the user types 'THE RULES ARE', then the routine starting at line 4000 will display all of the rules on the screen.

Line 200: The program comes here on any regular word. Each of four variables is set to an initial value of 0. The variables are: J for Odd or Even, D for Double Letter, V for the Vowel counter, and C for the Consonant counter.

Line 210 uses the RIGHT$(Z$,1) function to get the leftmost (First) character of the string Z$ into the string variable F$.

Line 220 uses the RIGHT$ (Z$,1) function to get the rightmost (Last) character of the string Z$ into the string variable L$.

Line 230 uses the LEN(Z$) length function to get the length of the string Z$ into the numeric variable L.

Line 240 uses the INTeger function to get the integer portion of the length L, does a logical AND with the number 1, and if the result is not zero (which will be the case for any ODD number) sets the J variable to 1 to be used later in the program to determine if the string was an ODD or EVEN number of characters.

Line 250 simply tests for a single letter word that obviously could *not* have a double letter, and skips the next few lines on a single letter.

Line 260 sets up a FOR... NEXT loop to test for a double character. The MID$(Z$,K,1) portion of line 270 isolates one character in the word, the MID$ (Z$,K+1,1) isolates the next character in the word, and the IF function tests to see if these two consecutive characters are identical. If they are, it sets the double flag, D, to 1 for later testing; if not, it leaves the flag alone.

Lines 290 to 350 are a FOR...NEXT loop to count Vowels (A E I O U and Y) and Consonants (all non-vowels) in the word. Line 300 sets the V$ string variable equal to the next character in the word. Lines 310 and 320 test for the vowels and line 340 adds one (1) to the V counter for each vowel found. Line 330 adds one (1) to the C counter for any character that is not a vowel.

All variables that are going to be used for the various RULE tests below now have been calculated:

F$ is the First character
L$ is the Last character
J = 0 for an odd number of characters
 = 1 for an even number of characte
L = the number of characters
V = the number of vowels
C = the number of consonants
D = 0 for no double letters
 = 1 for double letters

Line 1000 branches to a subroutine based on the number of the rule in R. It will go to subroutine 1100 on rule 1, 1200 on rule 2, ..., 1500 on rule 5.

Line 1100 tests for odd or even number of characters. If J = 0, which means that the word is disliked, it goes to line 2000 to display the word in green; otherwise the word is liked and goes to line 2100 to display the word in red.

Line 1200 tests for the length to be greater than 5 and, if it is, goes to line 2000 to turn the word green; otherwise it goes to line 2100 to turn the word red.

Line 1300 tests for more consonants than vowels and, if there are, goes to line 2100 to turn the word green.

Line 1400 tests for the first letter of the word to be earlier in the alphabet than the last letter. If so, it goes to line 2100 to turn the word red.

Line 1500 tests for double letters. If there are double letters, it turns the word red.

Line 2000 moves the cursor back one line to the typed word, writes two spaces to advance the cursor and remove the question mark, then changes the color to green.

Line 2010 prints the word (Z$) in whatever color has been set, then changes the color back to blue.

Line 2020 RETURNs, ends the subroutine, and returns to line 10000 and the GOTO 120 statement.

Line 2100 moves the cursor back to the word line and changes the color to red.

Line 2110 goes to line 2010 to complete the output of the red word.

Lines 3000 to 3040 handle the response when the user types 'THE RULE IS'. Line 3010 goes to subroutine 3100, which prints the RULE. Lines 3020 and 3030 print an extra carriage return and the message 'PRESS RETURN TO CONTINUE'. Line 3040 waits for a RETURN from the keyboard and then RETURNs to the calling program at line 130.

Lines 4000 to 4050 handle the response when the user types 'THE RULES ARE'. Line 4010 PRINTs a clear screen character (represented by the reverse heart character), sets the color to black (represented by the solid black box), then the message 'WORD DETECTIVE RULES:', and then changes the color to blue (represented by the reversed arrow). The next PRINT provides an additional blank line on the display.

Line 4020 uses a FOR...NEXT loop to output each of the five rules. It first PRINTs the color to black (solid box), the message 'RULE -', and then outputs the current rule number, R.

Line 4030 goes to the subroutine at line 3110 to output the "I LOVE..." information, a blank line, and returns.

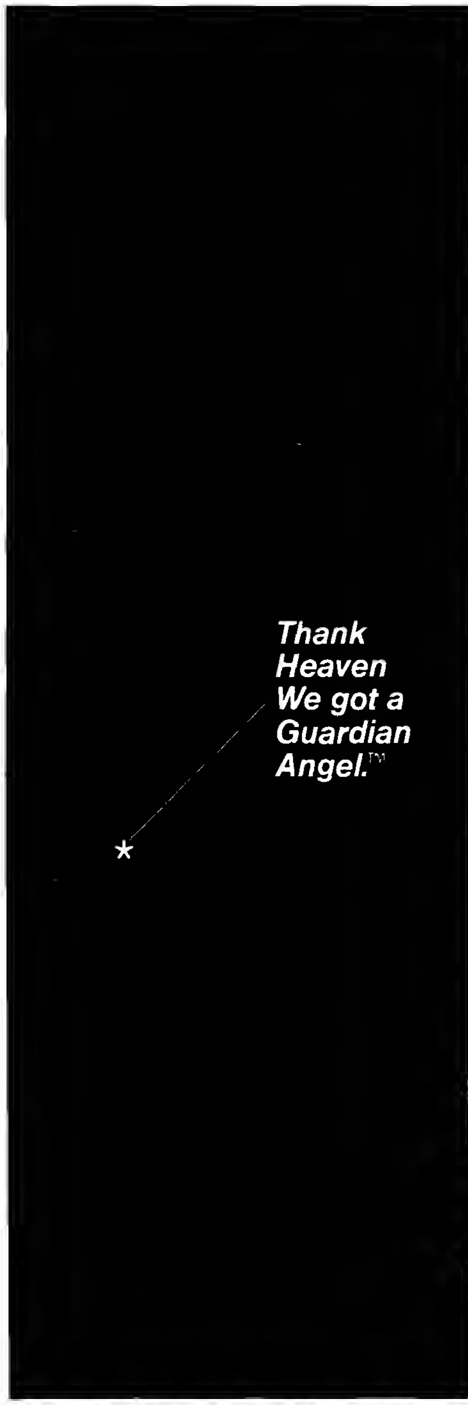
Line 4040 is the NEXT statement of the FOR...NEXT loop that began in line 4020. On R values of 1 to 4 this statement will go back to line 4020. On R equals 5 (the last value specified in the FOR loop in line 4020) the program will go to the next sequential line.

Line 4050 goes to line 110 to select a new random number for the next game.

That's all there is to it. Word Detective is a simple game and a simple program. If you study the program and fully understand it, then you are well on your way to understanding BASIC programming on the VIC-20.

MICRO™

The author may be contacted by writing to MICRO Magazine, P.O. Box 6502, Amherst, NH 03031.



HAVE YOU FLOWN YOUR ATARI TODAY?

Imagine yourself at the controls of a small, single-engine plane, 10,000 feet in the air, on your approach to the runway and safety. You're running low on fuel, but your instruments show that you're on the glide path, and lined up with the runway. It's a beautiful, sunny day, and you can see the airport in the distance, across the grassy fields. But the crosswind is tricky, and it will take all your skill to land safely. You're coming down now, and the runway is getting closer. A bit left, OK, now lower the power, fine, now put down the flaps. Pull the nose up a bit more, you're a little low. Watch the power! Don't stall. OK. Here comes the runway. You hear the squeal of tires on pavement, your pulse quickens, you' down, but watch it, you're pulli right! Brakes, brakes! Left mor You've stopped safely! Good jo The first real-time flight simulat for ATARI is now available frc MMG Micro Software. Written e tirely in machine languag there are four levels of difficult landings in clear or fog weather, landings with or w out instruments, and with without the real-time vie from the cockpit. **Fir Flight!** requires At 400/800, 24K, 1 joy stic and is offered on tape disk for the same su gested retail pri of $29.95.

MMG

F Flig is available your local dealer or d from **MMG Micro Software.** send check or money order to P.O. 131, Marlboro, N.J. 07746 or for Master Visa, and C.O.D. deliveries call **(201)431-3472.** Pl add $3.00 for postage and handling. New Jersey residents 6% sales tax.

Circle No. 46 Atari is a registered trademark of A

Lo-Res SHAPE Drawer

by Doug Denby

Although the Apple shape system is very good, it works only on the high-resolution screens. I have expanded the shape-drawing capabilities to include drawing a shape from a normal shape table on the low-resolution screen. This can be useful when examining shapes in detail, when using font tables to place messages on the screen in large banner style, when doing lo-res animation, or when you want to see a shape in a larger form or in one of 16 colors.

The assembly code is designed to work with either Applesoft or Integer BASIC. It is necessary for Integer BASIC to substitute the commented code (lines 60-62) for the beginning lines (31-56). If you want to use shapes with Integer BASIC, the Programmer's Aid ROM is necessary. If you are loading Integer BASIC into the language card (16K RAM Card) with the system master then you are also loading the programmer's aid ROM routines into place. Many Apple owners are unaware of these powerful routines. Integer runs faster than Applesoft (especially in graphics routines) because of simple number crunching. More programmers should be looking towards Integer BASIC as a programming language; the 64K RAM on the Apple //e will make it possible.

Back to the lo-res shape-drawing utility. I designed this routine to examine, in detail, shapes from standard shape tables. The shapes I wanted to see were font (character) tables and I knew they were designed to start at the top left corner of the character and proceed down the screen in a zig zag fashion. I located the origin of the shape two dots from the top left corner of the lo-res screen. This is done in lines 64-67. You may wish to change this. The demo program POKEs different numbers to these locations to move the shapes around the screen.

Because there is no method to check for scale and rotational factors, this routine runs much faster than the normal shape-drawing methods on the hi-res screens with either Integer or Applesoft BASIC. Animation could be performed effectively from BASIC on the lo-res screen by modifying the routine to start in a variable location that would be passed to the routine. (See the demo program.)

The different sizes of dots and their proportions on the lo-res screen as compared to the hi-res screen made it necessary to place two vertical lo-res dots for each hi-res dot in the shape to reduce shape distortion.

I did not include rotation or scale capabilities because they were not necessary. You can add them, but be careful of scale changes on the lo-res screen — the shape will already be quite large. I have made the screen wrap around as it does on the high-resolution screen to prevent range errors and to be able to see the whole shape if it is in fact too large for the 40 × 20 format of the lo-res screen.

Standard locations are used for the pointers to the shape table and on-the-fly shape pointer. Color is set with the normal COLOR= command in either BASIC.

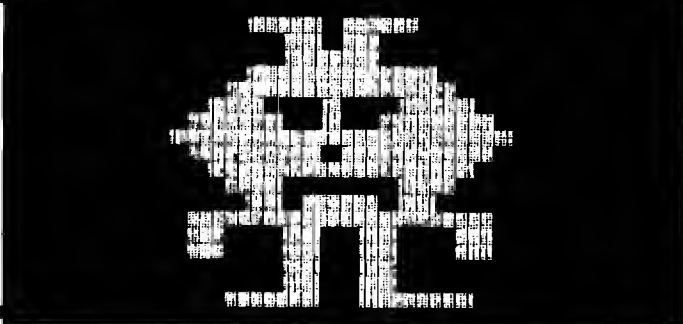
The program was written on the ORCA/M assembler and is liberally sprinkled with comments. I located it at $4000, but you can locate it where you think there will be free memory.

If you don't have an assembler, enter the monitor with a "CALL-151". Type "4000:" and enter the hex numbers in listing 3. It is not necessary to enter the line numbers after each "RETURN". Just type a ":" and continue. Be sure to leave a space between the hex numbers. Check your work often; a mistake requires retyping. Save the program with "BSAVE DENBY, A$4000, L$ AE". Do the same with listing 4, starting at $6000. Save it with "BSAVE SHAPES, A$6000, L$ 166."

To use the routine, BLOAD it and BLOAD the shape table. Set the Graphics screen and the COLOR=. POKE the location of the shape table into 232 and 233 ($E8 & $E9) as usual. These instructions work for either BASIC, but passing the particular shape in the table to the routine varies depending on which BASIC is being used.

With Applesoft BASIC, the shape number must be passed directly to this routine. This is done by POKEing the shape number into the first memory location past the start of the routine.

With Integer BASIC and the Programmer's Aid, the shape number is passed in the normal manner. See page 55 in the Programmer's Aid manual (Apple Part #A2L0011).



Listing 1

```
10  TEXT : HOME
20  REM   ************************
30  REM   * DEMO PROGRAM
40  REM   * FOR LO-RES SHAPE DRAWER
50  REM   * BY D. DENBY
60  REM   * COPYRIGHT BY MICRO INK
70  REM   * (C) MAY 1983
80  REM   ************************
90 D$ =  CHR$ (4)
100  PRINT D$"BLOAD DENBY"
110  PRINT D$"BLOAD SHAPES"
120  POKE 232,0: POKE 233,96: REM  SET SHAPE TABLE LOCATION
130  GR
140  PRINT "SHAPE 1 OR 2?": GET I
150 C = C + 2
160  IF C > 39 THEN C = C - 39
170  POKE 16420,0: REM   VERTICAL
180  IF I = 2 THEN  POKE 16420,20
190  POKE 16424,C: REM  HORIZOTAL
200  COLOR= 3
210  POKE 16385,I: REM  SHAP NUMBER
220  CALL 16384: REM  DRAW SHAPE
230  FOR J = 1 TO 5
240  GOSUB 320: REM  SNOWFLAKES?
250  NEXT J
260  POKE 16385,I
270  POKE 16420,0: POKE 16424,C
280  IF I = 2 THEN  POKE 16420,20
290  COLOR= 0: CALL 16384: REM  ERASE SHAPE
300  GOTO 150
310  END
320  POKE 16385,3
330 N1 =  INT (( RND (1) * 20) + 10)
340  POKE 16420,N1: REM  SET RANDOM VERT POSITION
350 N2 =  INT (( RND (1) * 40) + 1)
360  POKE 16424,N2: REM  RANDOM HORZ POSITION
370  COLOR= 3
380  CALL 16384: REM  DRAW SHAPE
390  POKE 16385,I: REM  SHAPE NUMBER
400  POKE 16420,0: POKE 16424,C: REM  HORZ AND VERT POSITION
410  IF I = 2 THEN  POKE 16420,20
420  COLOR= 7: CALL 16384: REM  REDRAW SHAPE TO COVER UP ANY ERASED PARTS
430  COLOR= 0: REM  BLACK
440  POKE 16420,N1
450  POKE 16424,N2
460  POKE 16385,3: REM  SHAPE NUMBER
470  CALL 16384: REM  ERASE SHAPE
480  HOME
490  RETURN
```

Listing 2

```
                     *      LOW-RES SHAPE DRAWING UTILITY
                     *      using Normal Shape Tables

                     *      Requires Programmer's Aid
                     *       if using Integer BASIC

                     *      No rotation or scale is taken
                     *      Screen wrap-around is in effect
                     *      On entry: Accumulator = Shape

                     *      Uses ROM Routines wherever possible

                     *      Equates:

                     GETPTR GEQU  $D361   PA entry to establish shape
                                          pointer
                     PLOT   GEQU  $F800   Monitor ROM Lo-Res plot routine

                     SHPTR  GEQU  $E8     FP Shape Table pointer
                     SHAPE  GEQU  $1A     Normal shape pointer position
                     TEMP   GEQU  $05     Temporary vector stroage
                     CODE   GEQU  TEMP+1  Current shape vector codes
                     VERT   GEQU  CODE+1  Vertical coordinate
                     HORZ   GEQU  VERT+1  Horizontal coordinate

                            ORG   $4000

4000 A900                   LDA   #$00    Dummy number

                     *      Applesoft needs to pass the shape number
                     *      to the routine, but note that the PA
                     *      has this function built-in.

4002 A8                     TAY           FP doesn't have a shape
4003 A5E8                   LDA   SHPTR   establishing routine that
4005 851A                   STA   SHAPE   is callable either, so
4007 A5E9                   LDA   SHPTR+1 this does the trick
4009 851B                   STA   SHAPE+1
400B 98                     TYA
400C A200                   LDX   #$00
400E 0A                     ASL   A
400F 9003                   BCC   NEXT1
4011 E61B                   INC   SHAPE+1
4013 18                     CLC
4014 A8              NEXT1  TAY
4015 B11A                   LDA   (SHAPE),Y
4017 65E8                   ADC   SHPTR
4019 AA                     TAX
401A C8                     INY
401B 811A                   LDA   (SHAPE),Y
401D 65E9                   ADC   SHPTR+1
401F 851B                   STA   SHAPE+1
4021 861A                   STX   SHAPE

                     **********         If using Integer BASIC then the
                     *      JSR   GETPTR    PA has a built-in routine
                     *      STX   SHAPE     for establishing shape
                     *      STY   SHAPE+1   use these lines instead

4023 A902                   LDA   #$02    Set starting position on screen
4025 8507                   STA   VERT
4027 A902                   LDA   #$02
4029 8508                   STA   HORZ
402B A000            LOOP   LDY   #$00    Get coded byte of vectors
402D B11A                   LDA   (SHAPE),Y
402F F07C                   BEQ   RETURN  If last vector, return
4031 E61A                   INC   SHAPE   Reset shape pointer
4033 D002                   BNE   NEXT
4035 E61B                   INC   SHAPE+1
4037 8506            NEXT   STA   CODE    Save the vectors
4039 8405                   STY   TEMP    Clear the current vector
403B 205740                 JSR   MARK    Decode & perform 1st vector
403E 205740                 JSR   MARK    Decode & perform 2nd vector
4041 A900                   LDA   #$00    3rd vector cannot be
4043 8505                   STA   TEMP    plotted or moved up,
4045 4606                   LSR   CODE    so is prefixed differently
4047 6605                   ROR   TEMP
4049 4606                   LSR   CODE
404B 6605                   ROR   TEMP
404D A505                   LDA   TEMP
404F F0DA                   BEQ   LOOP
4051 207240                 JSR   NOMARK
4054 4C2840                 JMP   LOOP

                     *      Main decode and move routine

4057 4606            MARK   LSR   CODE    Transfer move code to storage
4059 6605                   ROR   TEMP
405B 4606                   LSR   CODE
405D 6605                   ROR   TEMP
405F 4606                   LSR   CODE    Get plot command
4061 900F                   BCC   NOMARK  Plot only if told to
4063 A507                   LDA   VERT
4065 A408                   LDY   HORZ
4067 2000F8                 JSR   PLOT
406A A507                   LDA   VERT
406C 18                     CLC
406D 6901                   ADC   #$01
406F 2000F8                 JSR   PLOT
4072 0605            NOMARK ASL   TEMP    2 bits determine direction
4074 901C                   BCC   RIGHT
4076 0605                   ASL   TEMP
4078 9009                   BCC   DOWN
```

(continued)

Listing 2 *(continued)*

```
407A C608          DEC   HORZ
407C 102F          BPL   RETURN
407E A927          LDA   #$27
4080 8508          STA   HORZ
40B2 60            RTS

4083 E607   DOWN   INC   VERT    Takes 2 Lo-Res vertical dots
40B5 E607          INC   VERT    to match a Hi-Res dot
4087 A927          LDA   #$27
4089 C507          CMP   VERT
40BB B020          BCS   RETURN
408D A900          LDA   #$00
408F 8507          STA   VERT
4091 60            RTS

4092 0605   RIGHT  ASL   TEMP
4094 900D          BCC   UP
4096 E608          INC   HORZ
409B A927          LDA   #$27
409A C50B          CMP   HORZ
409C B00F          BCS   RETURN
409E A900          LDA   #$00
40A0 850B          STA   HORZ
40A2 60            RTS

40A3 C607   UP     DEC   VERT
40A5 C607          DEC   VERT
40A7 1004          BPL   RETURN
40A9 A927          LDA   #$27
40AB B507          STA   VERT
40AD 60     RETURN RTS
                   END
```

Listing 3

```
4000- A9 00 A8 A5 E8 85 1A A5
400B- E9 B5 1B 98 A2 00 0A 90
4010- 03 E6 1B 18 A8 B1 1A 65
4018- EB AA C8 B1 1A 65 E9 85
4020- 1B 86 1A A9 02 85 07 A9
4028- 02 B5 0B A0 00 B1 1A F0
4030- 7C E6 1A D0 02 E6 1B 85
403B- 06 B4 05 20 57 40 20 57
4040- 40 A9 00 85 05 46 06 66
4048- 05 46 06 66 05 A5 05 F0
4050- DA 20 72 40 4C 2B 40 46
4058- 06 66 05 46 06 66 05 46
4060- 06 90 0F A5 07 A4 08 20
4068- 00 FB A5 07 18 69 01 20
4070- 00 F8 06 05 90 1C 06 05
4078- 90 09 C6 08 10 2F A9 27
4080- 85 08 60 E6 07 E6 07 A9
408B- 27 C5 07 B0 20 A9 00 B5
4090- 07 60 06 05 90 0D E6 08
4098- A9 27 C5 08 B0 0F A9 00
40A0- 85 0B 60 C6 07 C6 07 10
40A8- 04 A9 27 85 07 60 00 00
```

Listing 4

```
6000- 03 00 10 00 B3 00 56 01
6008- F9 01 9C 02 3F 03 E2 03
6010- 09 09 2D 2D 29 2D 0D 09
6018- 11 1B 1B 1B 3B 1F 3F 1B
6020- 1B 13 09 09 09 2D 2D 0D
602B- 09 09 11 1B 1B 3F 3F 3F
6030- 3F 3F 1B 13 09 29 2D 2D
603B- 2D 2D 2D 0D 11 1B 3F 3F
6040- 1B 3B 1B 3B 3F 13 29 2D
6048- 2D 09 0D 29 2D 2D 15 3F
6050- 3F 3F 3F 3F 3F 3F 3F 37
6058- 29 2D 2D 2D 29 2D 2D 2D
6060- 15 1B 3F 3F 3F 3F 3F 3F
6068- 3F 13 09 29 2D 09 09 09
6070- 2D 0D 11 1B 1B 3F 3B 3F
6078- 1F 3F 1B 13 29 2D 2D 2D
60B0- 2D 2D 2D 2D 15 3B 1F 1B
608B- 3F 1B 3F 1B 3B 17 29 0D
6090- 09 2D 09 2D 09 29 15 1B
6098- 1B 1B 3F 1B 3F 1B 1B 13
60A0- 09 09 09 2D 09 2D 09 09
60AB- 11 1B 3F 3F 3F 1B 3F 3F
60B0- 1F 13 00 09 09 09 09 09
60B8- 29 2D 2D 2D 0D 09 09 09
60C0- 09 09 11 1B 1B 1B 1B 1B
60CB- 3F 3F 3F 3F 3F 3F 3F 1F
60D0- 1B 1B 13 09 09 29 2D 29
60DB- 29 29 29 29 29 29 2D 0D
60E0- 09 09 11 1B 3B 3F 3F 3F
60E8- 3F 3F 3F 3F 3F 3F 3F 3F
60F0- 3F 1F 13 29 2D 0D 0D 0D
60FB- 0D 0D 0D 0D 0D 0D 0D 2D
6100- 2D 2D 15 1B 3B 3F 3F 3F
6108- 3F 3F 3F 3F 3F 3F 3F 3F
6110- 3F 1F 13 29 2D 2D 2D 2D
6118- 2D 2D 2D 2D 2D 2D 2D 0D
6120- 09 09 11 1B 1B 1B 1B 1B
6128- 3B 3F 3F 3F 3F 3F 3F 1F
6130- 1F 1B 13 29 2D 0D 09 0D
6138- 09 09 09 09 29 09 09 09
6140- 09 09 11 1B 1B 1B 1B 1B
6148- 3B 3F 1B 1B 1B 1B 3F 1F
6150- 1B 1B 13 00 13 00 36 3F
6158- 24 27 2C 25 3C 2C 2D 2E
6160- 2E 3E 37 2E 3E 00 00 00
```

You may contact the author at 137 Main St., Unionville, Ontario, Canada L3R 2G6.

MICRO

A UNIX-Like Operating System for 6809 Microprocessors

Part II

by Stephen L. Childress

The modularization of the I/O system allows OS-9 to enhance the standard I/O at run time, not assembly or patch time. Device names and addresses are not fixed by the operating system but, rather, the program may attempt I/O to any device name. Of course, an appropriate module of that name must be loaded and ready to go at that time. Device names, driver procedures, register addresses, peripheral idiosyncracies, etc., are stated *outside* the core of the operating system and may be extended with ease. A device driver and descriptor for any data acquisition device may be loaded and accessed by programs using these techniques. In fact, should RBF and SCF fail to meet the needs of some device, an entirely new manager may be loaded and used alongside RBF and SCF. An example of this might be a 9-track tape drive that is not random-block oriented but is more than a character-oriented device. Perhaps an SBF (sequential block manager) module would be best:

This named-module concept is quite different from the old-school conventions, and it takes a while for the merits of the scheme to become apparent. The key point is that the modular software concept avoids considering where a piece of code is in memory until run time (not design time, assembly time, or link-load time). Although the name-to-address conversion takes a while, the module look-up directory makes the time acceptable.

You may wonder how modules can be expected to float around in memory. Won't machine instructions like JMP and JSR become confused? They would on an 8080 micro, but the 6809 uses relative addressing for branching, including long-distance branches and subroutine calls. Relative addressing is permitted on all instructions including those that access constants like numbers and strings. Since programs are *assigned* memory for variables (RAM) at run-time when invoked by KERNAL rather than the old way (assembly time), the four 16-bit index registers of the 6809 access data workspace without the programmer knowing where it will be.

After years of working the old way, these ideas took some time to register with me. Look at the tiny assembly-language program for OS-9 in figure 1. It should help you understand how position-independent module code works.

(Continued on next page)

```
* 6809's registers thus:
* the U index register points to my assigned RAM
* the S stack pointer points to the end of my assigned RAM
* the DP or base page register is set for same as U
* the Y register points to the user's command line, if any for
              picking up his desired actions.
* the D register has command line length
* the X index register points to bottom of stack
*    The mod statement's "mem" parameter specifies RAM size
*
*    In terms of I/O, "path" 0 is already opened to the user's
* keyboard, whatever that may be. Paths 1 and 2 are open to the
* display and error device, usually the user's screen. These paths may
* have been rerouted to other devices or files by the user's command
* line syntax, but that's not the concern of this code.
*
*
start   NOP                  do nothing
        LDA  #3              3 into A register
        STA  VARA,U          set variable's value
        LDD  #32767          big number in the D register
        STD  VARB,U          place it in VARB
        LBSR subr            call a possibly distant internal subroutine
* Say hello to user via his CRT on I/O path #1
        LEAX MSG,PCR       GET RUN-TIME ADDR OF ASCII in X register
        LDAS #1            I/O path 1 will be used
        LDY #MSGSZ         message size
        OS-9 I$WRLN        write ASCII on path 1
* "OS-9" is an SWI2 instruction to call the KERNAL, whatever that is
        BCC okexit         if no error, branch
        nop                do something about the error
okexit os9 I$exit          this is a SWI2 to exit to KERNAL
subr    RTS                subroutine does nothing
MSG     FCC "Hello World!"
        FCB $D             carriage return
MSGSZ EQU *-MSG            message size
        emod         tells assembler to emit module CRC
last    equ *        size of this module
        end
```

The code in this module is completely position-independent; i.e., it may be placed anywhere in memory and executed without link-loading or other preparatory adjustments, thanks to the 6809's powerful addressing. Look at the listing for a moment to understand how the code becomes modular and position-independent. You see the familiar ORG (origin) directive, but it means something quite different in the OS-9 system: ORG 0 sets up a "data section" of the program, which contains only memory reservations (RMBs).

The symbols VARA and VARB just after the ORG statement take on the values of zero and one, respectively; RMB produces no code. These numbers (actually the symbols VARA and VARB) will be used as offsets into the data storage area, whose exact address is not known at the moment. Remember, OS-9 assigns a data area (RAM region) at run time.

Now the MOD statement creates a module header, which contains all of the information needed to determine the module's name and attributes. The MOD statement introduces the "pure-code" part of a module, which should not contain storage for variables. When the KERNAL activates this module, the 6809's registers are set up as described in the listing's comments.

How do you address the variables? The 6809's U (index) register contains the address of the data area you will be using at the time the program runs. By now it should be obvious that each module may be given (by KERNAL) a unique piece of RAM for its variables. In this module, MEM in the header states that the module needs 203 hex bytes, minimum, of data area. The KERNAL's memory-management routine simply locates 203 bytes (actually, memory is handed out in multiples of 256 bytes) of unused memory and places that address in the U register before running the module.

Now consider the stack for the module; since the 6809 has a general 16-bit stack pointer register (S), the KERNAL simply sets S to the address of the *end* of the data area for the module. Thus, the module gets a private stack, unique from other modules' stack space. Why separate stacks? For time sharing. There may be several modules active simultaneously, each contending for CPU time as handed out by the KERNAL's scheduler. To switch modules, the KERNAL merely preserves the currently executing module's registers, loads up the new module's registers, and executes the new module starting where it last was when its time slice expired (due to a clock interrupt). Thus, time sharing (or multi programming) is made practical on even a small computer with only 64K. Each program uses just enough memory for its modules' code and a second region of memory for as many variables as needed. Therefore, many small modules or a fewer number of large modules may be run simultaneously.

Look again at the code in listing 1 to understand how to avoid ever using absolute memory addresses. Now you know addressing of variables merely uses an index register to point to a data area. But what about constants, which are imbedded in a module's pure code? The 6809's relative addressing mode comes to the rescue. In the listing, program is to send an ASCII message string to the user; it's labelled MSG, but remember that the location of the module in memory is not known until run time. The old way of getting the address of MSG is: LDX #MSG (or equivalent 8080/Z-80 code, etc.), which would place the value of MSG in the X register. But this is just the address shown in the listing, not the address at which MSG falls according to the module's address in memory. So the 6809's LEAX (Load Effective Address into X) instruction is used instead of LDX. The LEAX instruction contains (in bytes(s) following the opcode) the distance from the instruction to the label (MSG). When the LEAX instruction is executed, the 6809 chip does the following:

1. Gets the distance part of the LEAX instruction, which follows the opcode in one or two bytes.

2. Adds that distance value (which may be negative) to the current PC (program counter).
3. Places the result in the X register (16 bits).

After step 3, the X register has the actual memory address of MSG ready to ship off in a call for I/O to OS-9 (the OS-9 I$WRLN statement). The load effective address mode in the 6809 (we looked only at the PCR, PC-relative case) allows actual, absolute memory addresses to be determined easily at run time and ignored at assembly time.

Here are a couple of other key points shown in the listing: in order to be unaware of the addresses of the KERNAL's support for I/O calls, etc., all OS-9 system calls are made using the "OS-9" assembler directive. This merely produces a SW12 (software interrupt, trap, or whatever you wish to call it), which uses an address vector that the KERNAL has set up. The desired KERNAL action is stated by placing a code in the byte following the SWI2 instruction. In the listing example, the I$WRLN and F$EXIT codes are used. There are several dozen KERNAL function codes upon which to draw, each using the techniques shown for passing parameters in the registers. The point is that the KERNAL itself is a module like all the others and its address may be ignored.

Finally, the I/O performed (I$WRLN) by the module was done using a "path" number passed in a register. Like Unix, OS-9 programs are assigned three I/O paths at the time they are run. These are called the "standard paths" since few programs need more than three I/O "channels." When the user (or some other program) causes a module to be executed, the KERNAL sets up these three paths in a careful manner. Path zero is called the "standard input" and, for the case where a user invoked the module from a terminal, is associated with the terminal's keyboard. Paths one and two are the "standard output" and "standard error output," normally the user's terminal screen (or paper). The actual devices on the other end of these I/O paths are not known to a module (but may be determined if needed). Delightfully simple, a program merely reads on path 0 and writes on paths 1 and 2, completely ignoring what device is there.

Thinking back to the OS-9 block diagram shown in part I, the user's terminal often is affiliated with the SCF and ACIA modules that are handling I/O for the module. Each module activated by KERNAL is given paths 0, 1, and 2 for standard I/O, though path 0 in module A may be a different device than path 0 in module B. None the less, the modules are unconcerned with what's on the other end of the paths. Additional I/O paths to files or devices may be established at run time by any program.

Incarnations

Let's step back now and look at how these ideas all blend for the user's benefit. As an example, consider that two terminals are attached to the computer and each person wants to run BASIC. BASIC (BASIC09 for OS-9) is merely a large module that has all of the module conventions shown for the simple example. After booting up the system, the primary terminal is activated by automatically running the SHELL program module with the three standard I/O paths set up for the primary terminal, TERM. The user sitting at TERM's keyboard types in a command to invoke module SHELL with the I/O paths set for the second terminal, T1. Invoking a program causes the KERNAL to create a new "process". The time-sharing scheduler of the KERNAL gives each process the CPU for a time slice, then switches to another process. In this example there are two processes bidding for CPU time, both of which happen to be running the same program module, SHELL. SHELL, like any other module, is pure code and its variables are in a data space assigned by the KERNAL. The user at the TERM terminal and the user at the T1 terminal are, to OS-9, two separate processes, each having an assigned data area. Both processes are running the SHELL module's code; the same code is used for both processes, not two copies. As these two processes alternately get CPU time, they run sections of the code in SHELL, though not in lock-step. When a clock interrupt causes the KERNAL to switch processes, the PC for the interrupted process is one of the items preserved for recall when the process is later reactivated. Thus, the two processes running SHELL each march through the code in SHELL but have independent data and stack areas as well as I/O devices.

Since the two terminals are both running SHELL now, each user may instruct SHELL (*via* the keyboard) to run BASIC. The KERNAL dutifully handles the first user's request to execute module "BASIC09", copies it from disk to memory, and activates the BASIC09 module with a data area and the I/O devices for that user. When that user's (process) time slice expires (typically 1/10 of a second), the other user's process comes up and his request for BASIC is handed to the KERNAL. KERNAL discovers that BASIC09 is already in memory, so it simply activates BASIC09 a second time with the other user's data area and I/O devices. Now two people are sharing one copy of BASIC, each with private data areas and I/O devices. To emphasize that only one copy of the pure code within BASIC09 is present you could say that there are two incarnations of BASIC09 occurring. Time sharing makes each user feel as though he has his own CPU.

The time-sharing and Unix-like schemes I have shown here have been available previously only in minicomputer systems and presumed to be far beyond the means of the microcomputer. But they do work in OS-9 because:

1. OS-9 hands out memory in 256-byte chunks to minimize waste.
2. All code is "pure" so that you never need two copies of the same program in memory.
3. The 6809's addressing modes make handling separate data and program-code areas trivial.
4. All code is position-independent so that it may run anywhere without awkward and slow relocation.
5. The Unix I/O path philosophy allows programs to ignore exactly what devices it is working with. When required, a program can determine the device characteristics (file *versus* CRT, CRT controls, etc.) for special applications like screen forms and, of course, open disk files on other paths.

All This on a 64K Micro?

Study the features I've discussed to see whether or not they really are practical on a 64K computer (OS-9 also supports the SS50 machines with 20-bit bus addresses for 1Mb memories). For example: if you have one user who runs the LIST utility to copy a file to a printer and he/she simultaneously runs another program to edit a text file, these activities require:

LIST - 256 bytes of code, 512 bytes of data area.
EDIT - 8K bytes of code, 16K bytes of data area.

This is 8192+16384+256+512 of memory = 25,344 bytes. There is ample room for this, considering the size of the system software (KERNAL, IOMAN, et. al., as shown in the MDIR listing in part I). In fact, a second person may share EDIT with a new data area, or several more incarnations of LIST could run if other printers or devices are required. Since LIST is running concurrently, you have a print spool going without a $400 spool box.

The physical use of memory, in general terms, would be (approximately):

Mem Area	Utilization
A000-FFFF	OS-9 system code modules: KERNAL, I/O drivers, etc.
BF00-BFFF	Code module LIST, 0.25K
9F00-BEFF	Code module EDIT, 8K
5700-9EFF	(unused memory)
0600-46FF	Data area for EDIT, 16K
0400-05FF	Data area for LIST, 0.5K
0000-03FF	Data area for system modules

As you can see, there is ample unused memory for other concurrent programs (perhaps other users) to claim. Also, you could use a program larger than EDIT and still have room to spare.

In another instance, there might be two users, each running BASIC with relatively small programs (BASIC09 is an interactive compiler so program storage requirements are small). In this case, there would be a 21K area for BASIC and, say, two 8K areas for programs, for a total of 37K. If the collection of I/O drivers and other system software in use at the time were, say, 24K, the two-user BASIC case requires 24 + 37K = 61 of the 64K. This is a tight fit, but each user can run a sophisticated program in 8K of program space (about 250 lines of code, which is about a 5-page listing of BASIC code). If one of those users is running a program with large data requirements, such as big arrays, both users might not be able to use the system concurrently. Considering the highly sophisticated capabilities of BASIC09, it is indeed practical to run two users for many jobs on a small 64K computer. Clearly, running smaller programs like editors, assemblers, print-listers, and applications programs is much less demanding.

Interrupts — Immoral for Micros?

Surprisingly, few micros use interrupts, probably because of manufacturers' concerns about irregular schemes used by some peripheral vendors. Microware took a rather bold step and enabled interrupts within OS-9. With the reentrant module code, this turns out to be quite easy. But the clincher is that the serial I/O driver (ACIA) is designed expressly to handle CRT input and output *via* interrupts. For years the I/O boards have tied the interrupt request from the I/O chips (UARTs) to the bus IRQ line, but the operating systems never used interrupts! In OS-9, the ACIA driver accepts interrupts from the keyboard(s) and takes in characters when you type them. The keystrokes are buffered (up to 80 or more) by ACIA for each terminal. When the application program has time to read from the keyboard, ACIA delivers the characters thus far accumulated or, if none are buffered up, waits for more to arrive. The program is completely unaware of all this and may call for characters by ones or by lines. For output, ACIA accepts characters from a program and sends out one per interrupt to the terminal. If the program gets ahead of the device, in order to catch up ACIA puts the program to sleep for a short time. These same techniques may be done for parallel interfaces *via* the standard PIA driver.

The interrupt-driven I/O makes a profound difference in the usefulness of the system. In the SS50 machines, the serial cards contain the MC6850 UART, which has an interrupt request pin that is simply tied to the wire-or bus IRQ line — there are no vectors in hardware or other exotic conditions. How does OS-9 decide who is interrupting? Simple; each I/O driver, at startup time, tells the I/O manager (IOMAN) that it has devices that will interrupt, and states the address of each device's status register and which bit signifies a "done" condition. The interrupt sequence then goes like this:

1. Device has or needs data and sets IRQ on the bus true.
2. The 6809 does an interrupt sequence, stacking all of the registers and vectoring to a routine in IOMAN.
3. IOMAN goes through the list of devices, which it now knows may interrupt, and finds some device whose "done" bit is true.
4. IOMAN then branches to that device's I/O driver, say ACIA, passing

(Continued on page 91)

the address of the data area for that device (which was set up when the device was made known to OS-9).

5. The driver looks at the device's status register to decide what to do: input, output, error condition, etc.
6. The driver then tells IOMAN to return to the interrupted program.
7. IOMAN and KERNAL decide either return to the interrupted program or perhaps to switch processes.

Essentially IOMAN and the KERNAL make an I/O driver believe that the machine has a fancy vectored (and prioritized) interrupt system. Again, this simplifies the drivers. The ACIA driver does everything "right". For example, your keystrokes are accepted but not echoed back to the screen until some program actually reads the characters. This keeps the screen in order regardless of whether the user is ahead or behind the program's state. Also, if you fill up the keyboard buffer, ACIA sends a beep to the terminal to tell you to wait a moment. The usual "stop scroll," "abort program," "no echo," and other special keys are provided for also.

Many drivers may be written to use the periodic clock interrupts to poll the device status at some 100 times per second. This is convenient for controlling devices that cannot, due to the interface design, produce interrupts. But what about an old disk interface that uses programmed I/O (not DMA)? The driver for these disks will hog a lot of CPU time (one or two revolutions of the disk at 300 or so RPM) but, happily, interrupts may be left on while doing the long, slow seek; unless the user is blazing away on his keyboard, you don't suffer too much. And DMA-based disk controllers are the norm now, with many fancy LSI chips for DMA and disk control. The controllers for Winchester hard disks are so smart that the CPU time for servicing these is much less than for a floppy.

You can see that many of the features of the modern minicomputer operating system software are indeed feasible for even 8-bit microcomputers. OS-9 will run on an Apple II or a $3500 SS50 machine. At the time this was written, there was well-founded speculation that Tandy will offer OS-9 for the Color Computer. And why not?

You may contact the author at 31220 La Baya Dr., Suite 110, Westlake Village, CA 91362.

Player Utilities Package for APX fig-FORTH

The Player Utilities Package is a collection of FORTH words designed to manipulate the Players in Atari Player/Missile Graphics. The Package allows the user to define and move Player patterns easily on the video screen. An additional set of FORTH words demonstrates the capabilities of the Player Utilities Package.

by Mike Dougherty

Player/Missile (P/M) graphics of the Atari 800 allow many graphic special effects to be used in applications. Whether the P/M graphics are used to implement a fast-paced game or to provide the LOGO-style turtle for educational software, the graphics feature is an important programming tool. Unfortunately, APX fig-FORTH does not include any words to use the Atari P/M graphics feature. The following FORTH application, Player Utilities Package, is a set of FORTH words designed to control the Players of P/M Graphics. (For a description of Player/Missile Graphics refer to "Atari 800 Player/Missile Graphics," MICRO (44:9) and the *Atari Hardware Reference Manual.)* These FORTH words are not meant to be optimal in any specific sense; they should be used only as a starting point. As applications require P/M graphics, this utility may be tailored to fit as needed.

Enter screens 30 through 51 as listed. Note that screens 41 and 46 contain only the "next-screen" word '-->'. This is to allow room for your own utilities and enhancements. To enter the screens, use either of the line editors supplied with APX fig-FORTH, or use the screen editor I presented in the February '83 issue ("EDIT: An ATARI FORTH Screen-Oriented Editor," MICRO 57:47).

Once you have stored the screens on disk, type '30 LOAD' and RETURN to add the player-utility words to your FORTH vocabulary.

A Demonstration

To demonstrate the capabilities of the Player Utilities, the FORTH words in screens 54-62 are included also. These words define four Player patterns and four types of Player motion. DEMO executes the demonstration; it is not meant to be complete in any real sense but to simply demonstrate the Player Utilities Package application. DEMO allows the user to control the CROSS-HAIRS Player *via* joystick 1 (STICK(0) in BASIC) while the other three Players move on their own. A round is over whenever the CROSS-HAIRS touch another Player. This continues until a round terminates after a key has been pressed.

In general, the information describing a Player is pushed onto the stack followed by the required move parameter(s) prior to executing the Player movement word. For example, to move the TURTLE Player under control of joystick 1 (STICK(0) in BASIC):

TURTLE 0 JOY-MOVE

To move the DUCK Player 10 steps at random from its current position:

DUCK 10 RANDOM-MOVE

Player Data Structure

Each Player being displayed on the Atari video screen may be described by a set of attributes or characteristics. This attribute set consists of data elements such as the bit pattern being displayed, the location of that pattern on the screen, which Player number (0-3) is associated with the bit pattern, and so forth. To manage the set of Player attributes, a new data structure, PLAYER, was defined on screen 38. Graphically, this data structure is represented in figure 1. To reserve FORTH dictionary space for a Player data structure, the following steps are performed:

1. Push the following values onto the data stack:
   a. Each byte of the Player bit pattern, from the bottom of the image to the top.
   b. The number of bytes making up the Player pattern.
   c. The P/M Player number (0-3) for this Player.
2. Execute the defining word, PLAYER, to create a dictionary entry.
3. Follow PLAYER immediately with the name of the new Player data structure.

For example, the following will set up the Player data structure outlined in table 1:

```
HEX
49 2A 1C 3E 7F 7F
3E 5D 49 1C 1C 08      (bit pattern)
0C 02                  (size, Player#)
PLAYER TURTLE          (dict entry)
DECIMAL
```

Once defined, execution of the word TURTLE leaves the TURTLE parameter field address (pfa) on top of the stack. The pfa address points to the first memory location of the TURTLE Player data structure. By adding the correct offset to the pfa, the address of any Player attribute can be computed.

Since it is likely that the Player data structure will be modified from application to application, the Player Utilities Package references the attributes indirectly. FORTH screen 39 defines the common entry points into the Player data structure. For example, to push the current x location of TURTLE onto the stack, the following words would be executed:

TURTLE X@

The word TURTLE pushes the starting address of the TURTLE data structure (pfa) onto the data stack. The word X@ takes the pfa on top of the stack, computes the address of the x Player attribute, and replaces the top of the stack with the value of the x attribute. Similarly, the following places the top of the stack into the TURTLE's x attribute:

TURTLE X!

A FORTH word requiring the x value of TURTLE does not need to know where x is stored in the data structure. Further, if ever the position of x in the Player data structure must be moved, only X@ and X! need to be changed. (Naturally, any application in memory would have to be forgotten and re-LOADed.

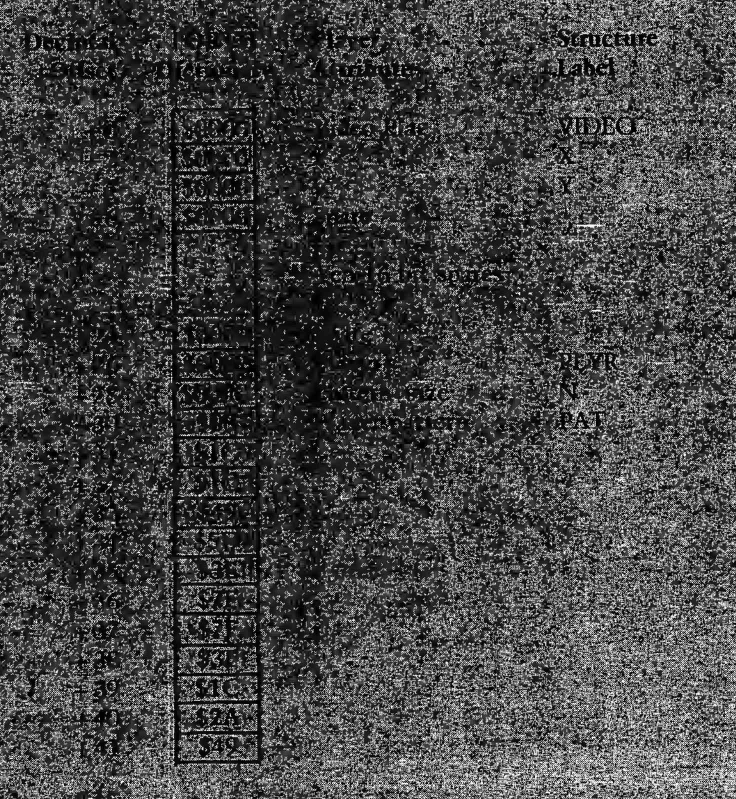
In addition to being modified easily, PLAYER defines ten unused words for immediate expansion. When the Player Utilities Package was written, I had several applications that required the extra space. For your own applications, PLAYER should be modified as necessary.

Player Movement

The Atari 800 Player is a narrow graphic image extending vertically from top to bottom on the video screen. The horizontal screen position of a Player is determined by a hardware horizontal position register. The

(Continued on next page)

Figure 1: TURTLE Player Data Structure after Initialization



Player Utilities Package requires:
Atari 800
Atari 810 Disk Drive
APX fig-FORTH
32K bytes of memory

horizontal position is changed by storing the 8-bit position (0-255) desired in the Player's horizontal register. The apparent vertical position of the Player is determined by the location of the Player bit pattern in P/M memory. A Player is moved up or down by moving the bit patterns. However, any bits remaining from the old image will still be displayed until they are explicitly erased with zeros. Thus, to move a Player an arbitrary amount the old Player bit pattern must be filled with zeros (erased), the Player bit pattern must be moved to the new Player memory position for vertical movement, and the Player horizontal position register must be set for horizontal movement. To avoid flicker, the first two steps should be performed as close together as possible.

The Player Utilities Package provides two FORTH words to move Players about the video screen. The word STEP-XY allows a Player to move relative to its current position, while MOVE-XY specifies an absolute destination. The x,y location of a Player is restricted only by the range of a FORTH 16-bit number. If a Player falls outside of its graphic memory, the Player VIDEO attribute is set to zero and the image is not displayed. However, the x,y position attributes do not depend upon whether or not the Player is displayed — the video screen acts as a window into the Player's space. Since both STEP-XY and MOVE-XY make extensive use of the Return Stack for parameter storage, they are coded as single FORTH words. Listing 1 shows a pseudo-code representation of STEP-XY.

STEP-XY expects the Player pfa, the relative step in the x direction, and the relative step in the y direction to be on the stack. For example:

```
TURTLE 5 - 7 STEP-XY
```

will move the Player TURTLE five steps to the right and seven steps upward. (As in normal graphics, the low addresses in P/M Memory represent the top of the video screen.)

In a similar manner, MOVE-XY expects the Player pfa, the final x position, and the final y position to be on the stack. For example:

```
TURTLE 128 60 MOVE-XY
```

moves the Player TURTLE near mid-screen for a double-line (46) playfield. (Since every television is adjusted slightly differently, specific Player positions will differ from set to set. For this reason Atari does not utilize the full video screens; instead, a generous margin is left for poorly adjusted televisions.)

Conclusion

The Atari Player/Graphics is an important feature of the Atari Personal Computer. Hopefully the Player Utilities Package will provide sufficient insight to allow serious P/M Graphics applications to be developed in FORTH.

You may contact the author at 7659 West Fremont Ave., Littleton, CO 80123.

List 1

```
BEGIN STEP-XY
  The parameters for erasing the pattern are initialized;
  The y step increment is added to the current y value;
  A check is made to see if y may be displayed;
  The x step increment is added to the current x value;
  A check is made to see if x may be displayed;
  IF the Player x and y may be displayed on the screen;
    The display parameters are initialized;
    IF the Player is currently being displayed;
      Erase the old bit pattern;
    ELSE the Player is not displayed;
      Drop the erase parameters;
    ENDIF;
    Display the Player at the new location;
    Flag the Player as currently being displayed;
  ELSE the Player is out of display range;
    IF the Player is currently on the screen;
      Erase the old image;
    ELSE the Player was not on the screen;
      Drop the erase parameters;
    ENDIF;
    Flag the player as currently not being displayed;
  ENDIF;
END of STEP-XY.
```

***Table 1:* Player Utilities Package Glossary**

?COLLISON (bit offset — 1) ....Screen 45
Return a logical value indicating whether or not the "bit" in collision register "offset" has been set by the hardware. If set then the player associated with "offset" has collided with the player associated with "bit."

?VIDEO (value min max — value 1) ....Screen 47
Determine whether or not "value" is inclusively between the range of "min" and "max." If "value" is outside the limits, return a false logical flag; otherwise return a true logical flag. "Value" is returned as the second element of the stack regardless of the limit check.

CLEAR-MEM (—)..Screen 32
Clear the reserved Player/Missile graphics memory. To prevent disabling the Atari graphics by accidental execution of CLEAR-MEM, the P/M graphics is defined to be the 4K above what the OS thinks is top of memory. (See RESERVE-MEM)

CLEAR-P (pfa — to n) ....Screen 43
Set up the parameters required to ERASE an old pattern.

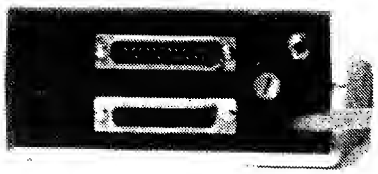
CLEAR-XY (pfa —).......................................Screen 43
Erase the Player whose data structure pfa is on the stack. The pfa is pushed onto the stack by using a defined pattern (see PLAYER).

(Continued on page 96)

ZANIM SYSTEMS

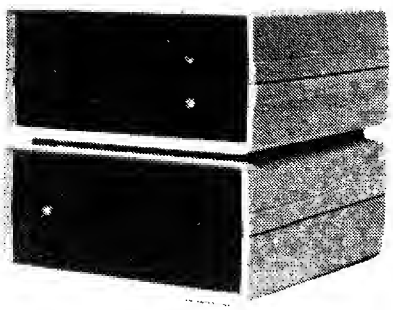
P.O. Box 4364
Flint, Michigan 48504
(313) 233-5731
(313) 233-3125

HOME CONTROL AT YOUR FINGERTIPS



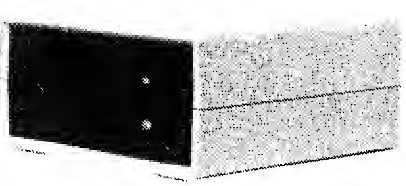
The ZCM-1 is the Master Control module that provides the interface between your computer and our line of Zanim Application Modules. Up to 15 Application Modules can be piggy-backed to the ZCM-1 Master Control module. The ZCM-1 is compatable with any standard RS-232 (serial) interface. A special Master Control module, the ZCM-1V is available for the VIC-20 and Commodore-64 computers.

*The ZCM-1V is available for VIC-20 and C-64 users.



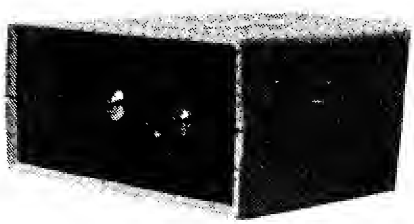
ZAM-1 is the home control interface module that provides a fully versatile computer controlled environment. ZAM-1 can control up to 256 different lamps and appliances in your home or business providing you with an effective and easy to implement energy management and electric control system. No special modifications are necessary to your building as all control signals are sent over your existing wiring. ZAM-1 can be programmed in BASIC or optional home control software is available. ZAM-1 requires one ZCM-1 Master Control module.

*The ZCM-1/ZCM-1V Master Control module is required to use the ZAM-1 Home Control module.



The ZAM-2 allows your computer to continuously monitor up to 15 different doors or windows around your home or business. ZAM-2 is a basic building block in a complete computer controlled home security system. With our ZAM-1 Home Control module, you can have a fully integrated security and environment control system. Upon an intrusion, your computer can take the action most appropriate, whether that is to ring an alarm bell, flash all the lights around your home, or dial the police.

*The ZCM-1/ZCM-1V Master Control module is required to use the ZAM-2 Security module.

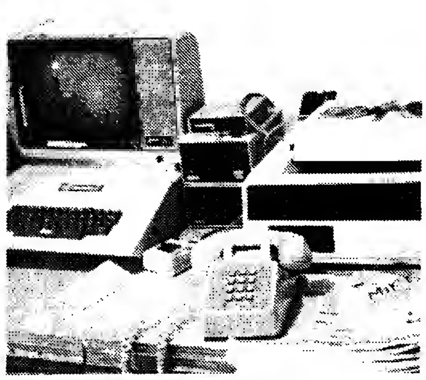


The ZAM-3 is a complete telephone answering and dialing system. It is capable of taking the phone off-hook and dialing a number under computer control or of answering the phone when it rings. With the ZAM-1 Home Control module and the ZAM-2 Security module, the ZAM-3 Phone Dialer module can be integrated into a complete home or business security/monitoring system. Applications include security, auto phone dialing, and computer-answering systems.

*Pulse dialing option is available as ZAM-3P.

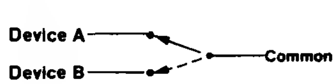
*The ZCM-1/ZCM-1V Master Control module is required to use the ZAM-3/ZAM-3P Phone Dialer module.

SERIAL OR PARALLEL (CENTRONICS) PORT SWITCHER



DOES YOUR COMPUTER LOOK LIKE THIS?

A PORT SWITCHER NOW AVAILABLE FOR YOUR COMPUTER (ZSW1)



Circle No. 54

P.O. BOX 4364
Flint, Michigan 48504
(313) 233-5731
(313) 233-3125

☐ Please send me more information or catalogue!

Name ______________________

Company ______________________

Address ______________________

City __________ State ________ Zip ________

What Make/Model Computer do you own?

____

APPLE 11 IS TRADEMARK OF APPLE COMPUTERS, INC.
CP/M IS REGISTERED TRADEMARK OF DIGITAL RESEARCH, INC.
TRS – 80 IS REGISTERED TRADEMARK OF TANDY CORP.
VIC - 20 AND COMMODORE - 64 ARE REGISTERED TRADEMARKS OF COMMODORE INC.

Table 1 *(continued)*

COLLISION-OFF (—) ....Screen 45
Reset the collision detection hardware for all players and missiles.

GET-XY (pfa — x y) ....Screen 42
Get the current x,y position of the player.

MOVE-XY (pfa x y —) ....Screen 50
Move the Player to the new x,y position. If the new position is in the Player's graphic memory, the Player will be displayed.

N! (n pfa —) ....Screen 39
Set the Player pattern size to "n".

N@ (pfa — n) ....Screen 39
Get the Player pattern size.

P/M-DEFAULT (playfield —) ....Screen 33
Set up the P/M graphics to "playfield" size (46 = double line or 62 = single line). The FORTH variables BYTES/PLAYER, P/M-OFFSET, and P/M-PLAYFIELD are initialized. In addition, each of the four Players are set to position (0,0) with normal size and gold color.

P/M-OFF (—) ....Screen 34
Disable the Player/Missile DMA.
Note: The Players are moved off the screen due to a timing problem with disabling the DMA. Occasionally, turning off the DMA leaves a "solid bar" of color where the Player was last displayed. By moving the image off the screen, the visual problem is avoided. I have noticed no "side effects" with this solution.

P/M-ON (—) ....Screen 34
Enable the Player/Missile graphics DMA.

P/M-SETUP (playfield —) ....Screen 36
Set up the Players to the "playfield" default value. This word initializes the P/M memory, sets the Player defaults, and enables the P/M DMA.

PAT@ (pfa — pattern-address) ....Screen 39
Get the pattern address of the Player.

PLAYER Compile: (Pn ... P1 n Player# —) ....Screen 38
Execute: (— pfa)
Define the Player defining word. To define a new Player, push the Player pattern onto the stack (bottom to top), followed by the number of bytes in the pattern, and the Atari Player number (0-3) for that pattern. Player builds the Player data structure required by the Player Utility words. For example:

```
255 129 129 255 4 1 PATTERN BOX
```

would create a 4-byte Player named BOX associated with Atari Player #1. Executing the defined pattern, BOX, would leave the pfa of the BOX data structure on the stack. This pfa is the primary Player argument in many of the Player Utility Package words.

PLYR! (player# pfa —) ....Screen 39
Associate the Player data structure with Atari "Player#."

PLYR@ (pfa — player#) ....Screen 39
Get the Player number from the data structure.

PLYR-COLOR (pfa color lum —) ....Screen 40
Set the Player color and luminance as in the Atari BASIC setcolor command. (A Player can have only one color.)

RESERVE-MEM (—).....................................Screen 32
Reset the Atari top-of-memory variable in location 106 (decimal) to reserve 4K bytes of memory for P/M graphics. Save the original memory value in MEM-TOP. This approach prevents conflict with the Atari OS.

RESTORE-MEM (—) ....Screen 35
Disable P/M graphics and reset the original memory pointer in location 106 (decimal).

SET-XY (pfa x y —) ....Screen 42
Set the Player to the absolute x,y position.

SHOW-H (*x* player # —) ....Screen 44
Set the horizontal position register of Player "player#" to "x."

SHOW-P (pfa — x player# from to n) ....Screen 44
Set up the parameters to display a Player on the screen.

SHOW-XY (pfa —) ....Screen 44
Display a Player on the screen. This word does not check for a valid x,y position or set the Player video flag. Use with caution in the direct mode.

STEP-XY (pfa incx incy —) ....Screen 48
Step the Player from its current position by the amount "incx" and "incy." This step is relative to the Player's current position.

VIDEO! (displayflag pfa —).............................Screen 39
Set the Player video flag. A value of zero means that the player is out of the Player display space and is not displayed. A value of one means that the Player is currently displayed on the screen.

(continued)

Table 1 *(continued)*

VIDEO@ (pfa — displayflag) .... Screen 39
Get the Player video flag.

X! (x pfa —) .... Screen 39
Set the Player x position.

X@ (pfa — x) .... Screen 39
Get the Player x position.

Y! (y pfa —) .... Screen 39
Set the Player y position.

Y@ (pfa — y) .... Screen 39
Get the Player y position.

Y-ADDR (y player# — addr) .... Screen 37
Compute the memory address of the Player pattern in P/M memory. This is the vertical (y) position of the Player bit pattern.

List 2

```
SCR # 30

( APX fig-Forth Player Graphics Utility Words                   )
(                                                                )
(          by  Mike Dougherty                                    )
(                                                                )
( These Forth words extend APX fig-Forth to allow easy user      )
( manipulation of the Atari 800 "Players" for educational        )
( applications, games, etc.                                      )
(                                                                )
( To compile utility words:      30 LOAD                         )
( To compile demonstration:      54 LOAD                         )
(                                                                )

-->


SCR # 31
( CONSTANTS AND VARIABLES )

53248  CONSTANT P/M-HORZ         ( P0 horizontal position reg )
53256  CONSTANT P/M-SIZE         ( P0 size register )
  704  CONSTANT P/M-COLOR        ( P0 color shadow register )
54279  CONSTANT P/M-PAGE         ( Pointer to P/M graphics page )
  559  CONSTANT P/M-PLAYFIELD    ( Playfield type )
53277  CONSTANT P/M-DMA          ( P/M DMA enable/disable reg )
53248  CONSTANT COLLISION        ( Base collision register )
106 C@ VARIABLE MEM-TOP          ( To save original mem size )
0      VARIABLE BYTES/PLAYER     ( For Player memory size )
0      VARIABLE P/M-OFFSET       ( Offset in P/M mem to P0 )

-->


SCR # 32
( ALLOCATE / CLEAR P/M MEMORY )

: RESERVE-MEM                    ( --- )
  106 C@ MEM-TOP !               ( Save original value )
  106 C@ 16 - 106 C!             ( Reserve 4K for P/M graphics )
  106 C@ P/M-PAGE C!             ( Point P/M hardware to mem )
  XGR ;                          ( Reset display in new memory )


: CLEAR-MEM                      ( --- )
  106 C@ 256 *                   ( P/M memory base address )
  4096                           ( Number of bytes )
  ERASE ;                        ( Fill with zeros )

-->
```

(Continued on page 100)

SAGE TECHNICAL BRIEFING

SYSTEM DESIGN, SAGE IV

1

Meet The New Giant Of The Microcomputer Industry. All 4.8 × 12.5 × 16.75" Of It.

The challenge was to create a computer having room for a megabyte of RAM, a built-in Winchester with floppy backup, and the ability to perform 2,000,000 instructions per second.

A small miracle, in other words.

And small is exactly what it turned out to be. In fact, the 16-bit Sage™ IV, including all of the above attributes, takes up less than ½ cubic foot.

What makes such a breakthrough possible? System design.

It took the latest 64K dynamic RAMs and the Motorola 68000 processor technology, plus Winchester technology. And it took a highly integrated, closely packed, low power, high speed design incorporating a proprietary bus.

Now the Sage™ IV is ready for you. Actually, you can choose from three different Sage™ IV models to meet your exact needs—configurations with a 5 megabyte Winchester plus 640K floppy right on up to a combination of four fixed or removable Winchesters plus one or two floppies (200 megabytes of disk capacity in all).

Because of the Sage™ IV's no-compromise system design you can load a 16K program in 1/10 second from Winchester disk.

What's more, there are over 120 sources for existing popular programs for the Sage™ IV. The incredible p-System operating system, standard on every Sage™ IV converts software that was originally written for 8-bit computers in Pascal, BASIC and Fortran. Optionally, CP/M-68K Modula, and Hyper-Forth are also available.

Better yet, our small miracles come with prices to match.

So give us a call or write today for more Sage™ IV information and the name of your nearest dealer.

Western United States
Sage Computer Technology,
35 North Edison Way, #4, Reno, NV 89502 (702) 322-6868

Eastern United States
Sage Computer Technology,
15 New England Executive Park
Suite 120, Burlington, MA 01803
(617) 229-6868

In UK
TDI LTD, 29 Alma Vale Road,
Clifton, Bristol BS8-2HL
Tel. (0272) 742796
Tx: 444 653 Advice G

In Germany
MM Computer, GmbH,
Hallwanger Str. 59, 8210 Prien
Tel: 08051/3074
Tx: 525 400 mmco-d

SAGE COMPUTER TECHNOLOGY™

Sage trademark of Sage Computer Technology
CP/M, trademark of Digital Research
Modula, trademark of Volition Systems
Hyper-Forth, trademark of Forthright Engineering
p-System, trademark of UC Regents

List 2 *(continued)*

```
SCR # 33
( DEFAULT PLAYER GRAPHICS VALUES )
: P/M-DEFAULT                        ( playflald --- )
  DUP 46 = IF                        ( If the typa ia 46 then )
    128                              ( Each player ia 128 bytes )
  ELSE                               ( Else forca to 62 default )
    DROP 62 256                      ( Each player la 256 bytes )
  ENDIF
  DUP BYTES/PLAYER !                 ( Save bytes per player )
  4 * P/M-OFFSET   !                 ( Player 0 atarts 4 down )
  P/M-PLAYFIELD   C!                 ( Save 62 or 46 as playflald )
  4 0 DO                             ( Defaulta for all 4 playars )
    0  P/M-HORZ  I + C!              ( Inlt to 0 position, offscreen)
    0  P/M-SIZE  1 + C!              ( Inlt aize to noraal )
    24 P/M-COLOR 1 + C!              ( Init color to gold: 2*16+8 )
  LOOP ;    -->

SCR # 34
( P/M GRAPHICS UTILITIES )
: P/M-ON                             ( --- )
  3 P/M-DMA C! ;                     ( Enabla P/M DMA to begin )
: P/M-OFF                            ( --- )
  4 0 DO                             ( Mova tha playera off screen )
    0 P/M-HORZ I + C!                ( By setting to horz pos 0 )
  LOOP                               ( alse ocasaional visual proba )
  0 P/M-DMA C! ;                     ( Disabla P/M DMA )
-->

SCR # 35
( P/M GRAPHICS UTILITIES )
: RESTORE-MEM                        ( --- )
  P/M-OFF                            ( Turn off P/M graphics DMA )
  MEM-TOP @ 106 C!                   ( Restore original mem limit )
  XGR ;                              ( Reset display in new memory )
-->

SCR # 36
( P/M UTILITIES )
: P/M-SETUP                          ( playfield --- )
  RESERVE-MEM                        ( Allocata mem at top of RAM )
  CLEAR-MEM                          ( Inlt to zero -- no players )
  P/M-DEFAULT                        ( Set to Player dafault valuaa )
  P/M-ON ;                           ( Enable tha P/M graphic DMA )
-->

SCR # 37
( P/M UTILITIES )
: Y-ADDR                             ( y player# --- addr )
  106 C@ 256 *                       ( Get basa addr for P/M mem )
  P/M-OFFSET @ +                     ( Add offset to Player 0 )
  SWAP BYTES/PLAYER @ *              ( Form offset for Player n )
  +                                  ( Add for addr of Playar n )
  + ;                                ( Add y displacament )
-->

SCR # 38
( DEFINE A PLAYER DATA STRUCTURE IN THE DICTIONARY )
: PLAYER
                                     ( At complla time: )
  <BUILDS                            ( Pn ... P1 n Player# --- )
    0 ,                              ( Init VIDEO dlsplay flag )
    0 , 0 ,                          ( Inlt x,y coordinates )
    10 0 DO  0 ,  LOOP               ( Room for future attributes )
    ,                                ( Put Player number in dlct )
    DUP ,                            ( Put pattern size ln dict )
    0 DO                             ( For each pattern value )
      C,                             ( Add byte in dict )
    LOOP
                                     ( At execution time: )
  DOES>                              ( --- pfa )
    ;           -->                  ( Slmply return pfa address )

SCR # 39
( COMMON PLAYER DATA STRUCTURE ENTRY POINTS )
: VIDEO@       @ ;                   ( pfa --- displayflag )
: VIDEO!       ! ;                   ( displayflag pfa --- )
: X@      2 + @ ;                    ( pfa --- x )
: X!      2 + ! ;                    ( x pfa --- )
```

(Continued on page 102)

List 2 *(continued)*

```
: Y@       4 + @ ;                  ( pfa --- y )
: Y!       4 + ! ;                  ( y pfa --- )
: PLYR@   26 + @ ;                  ( pfa --- player# )
: PLYR'   26 + ! ;                  ( player# pfa --- )
: N@      28 + @ ;                  ( pfa --- n )
: N!      28 + ! ;                  ( n pfa --- )
: PAT@    30 +   ;                  ( pfa --- pettern-address )
-->

SCR # 40
( PLAYER UTILITIES )
: PLYR-COLOR                        ( pfa color lum --- )
  SWAP 16 * +                       ( Fore color/lum byte )
  SWAP PLYR@                        ( Get the Player nueber )
  P/M-COLOR +                       ( Form Player color shedow reg )
  C! ;                              ( Set the shadow register )
-->

SCR # 41
-->

SCR # 42
( XY PLAYER UTILITIES )
: SET-XY                            ( pfa x y --- )
  ROT SWAP OVER                     ( x pfa y pfa --- )
  Y!                                ( Save y value in Pleyer )
  X! ;                              ( Seve x velue in Player )
: GET-XY                            ( pfa --- x y )
  DUP                               ( Copy Pleyer pfe )
  X@                                ( Get x Player value )
  SWAP Y@ ;                         ( Get y Player velue )
-->

SCR # 43
( ERASE PLAYER )
: CLEAR-P                           ( pfa --- "to" n )
  DUP N@ SWAP                       ( Put Player size on bottom )
  DUP Y@ SWAP                       ( Get y offset )
  PLYR@                             ( Get Player number )
  Y-ADDR                            ( Convert to screen "to" addr )
  SWAP  ;                           ( "to" n --- )
: CLEAR-XY                          ( pfa --- )
  CLEAR-P                           ( Get the "to" n erguments )
  ERASE ;                           ( Erase the Player image )
-->

SCR # 44
( OISPLAY A PLAYER )
: SHOW-P                            ( pfa --- x plr# "from" "to" n )
  DUP X@ SWAP    DUP PLYR@ SWAP     ( Get x, player# values )
  DUP PAT@ SWAP  OUP N@ SWAP        ( Set pattern addr, size )
  DUP Y@ SWAP    PLYR@  Y-ADDR      ( Set y address in P/M mem )
  SWAP ;                            ( x player# "from" "to" n --- )
: SHOW-H                            ( x player# --- )
  P/M-HORZ + C! ;                   ( Place value in horz reg )
: SHOW-XY                           ( pfa --- )
  SHOW-P                            ( Set up stack )
  CMOVE                             ( Move Pleyer pettern )
  SHOW-H ;                          ( Set up horz reg )
-->

SCR # 45
( COLLISION UTILITIES )
: ?COLLISION                        ( bit offset --- 1 )
  COLLISION + @                     ( Fore addr of collision reg )
  AND ;                             ( Mask to desired bit )
: COLLISION-OFF                     ( --- )
  0 COLLISION 30 + C! ;             ( Write 0 to dimable register )
-->

SCR # 46
-->

SCR # 47
( VIDEO LIMIT CHECK FOR PLAYER )
: ?VIDEO                            ( v min max --- v 1 )
  ROT SWAP OVER DUP >R              ( Save value: min v max v --- )
  < IF                              ( If v > max then ... )
    2DROP R> 0                      ( Return v, FALSE )
  ELSE                              ( Ok, v <= max )
    > IF                            ( If v < min then ... )
      R> 0                          ( Return v, FALSE )
    ELSE                            ( Ok, v >= min )
      R> 1                          ( Return v, TRUE )
    ENDIF
  ENOIF ;
-->

SCR # 48
( ADVANCE THE PLAYER )
: STEP-XY                           ( pfa incx incy --- )
  >R >R DUP DUP >R CLEAR-P          ( Set up erase parameters )
  R VIDEO@ R> R> R>                 ( Add video flg, restore stack )
```

```
   ROT SWAP OVER                     ( pfa to n v ix pfa iy pfa --- )
   Y@ +                              ( Form new y position )
   0 BYTES/PLAYER @ 1 - ?VIDEO       ( Check if y in graphic mem )
   0= >R                             ( Save result of test )
   OVER Y!                           ( Store new y regardless )
   SWAP OVER                         ( pfa to n v pfa incx pfa --- )
   X@ +                              ( Form new x position )
   0 255 ?VIDEO                      ( Check if x in graphic mem )
   0= >R                             ( Save result of test )
   OVER X!                           ( Store new x regardless )
 -->


SCR # 49
   R> R> + 0= IF                     ( Test for Player on screen )
     SHOW-P >R >R >R >R >R           ( Yes, set up/save show params )
     IF                              ( If previously on screen )
       ERASE                         ( Erase current image )
     ELSE
       2DROP                         ( Else ignore erase params )
     ENDIF
     R> R> R> R> R>                  ( Restore show params )
     CMOVE SHOW-H                    ( Move pattern, set horz reg )
     1 SWAP VIDEO!                   ( Set video flag to "on" )
   ELSE
     DROP                            ( Drop SHOW-P pfa arg )
     IF ERASE ELSE 2DROP ENDIF       ( If prev visible, clear pat )
     0 SWAP VIDEO!                   ( Set video flag to "off" )
   ENDIF ;
 -->

SCR # 50
 ( MOVE THE PLAYER )
 : MOVE-XY                           ( pfa x y --- )
   >R >R DUP DUP >R CLEAR-P          ( Set up erase parameters )
   R VIDEO@ R R> R> R>               ( pfa to n v pfa pfa x y --- )
   0 BYTES/PLAYER @ 1 - ?VIDEO       ( Check if y in graphic mem )
   0= >R                             ( Save VIDEO status )
   SWAP                              ( Get x value )
   0 255 ?VIDEO                      ( Check if x in graphic mem )
   0= >R                             ( Save VIDEO status )
   SWAP SET-XY                       ( Save x,y values in Player )
 -->

SCR # 51
   R> R> + 0= IF                     ( Test for Player on screen )
     SHOW-P >R >R >R >R >R           ( Yes, set up/save show params )
     IF                              ( If previously on screen )
       ERASE                         ( Erase old Player image )
     ELSE
       2DROP                         ( Else ignore erase params )
     ENDIF
     R> R> R> R> R>                  ( Restore show params )
     CMOVE SHOW-H                    ( Move pattern, set horz reg )
     1 SWAP VIDEO!                   ( Set video flag to "on" )
   ELSE
     DROP                            ( Drop SHOW-P pfa arg )
     IF ERASE ELSE 2DROP ENDIF       ( If prev visible, clear pat )
     0 SWAP VIDEO!                   ( Set video flag to "off" )
   ENDIF ;

SCR # 52
 ;S

SCR # 53
 ;S
```

List 3

```
SCR # 54
 ( DEMONSTRATE THE PLAYER GRAPHICS UTILITY )
 HEX                                 ( Define some Players )
 08 08 08 7F 08 08 08                07 00   PLAYER   CROSS-HAIRS
 7C FE FE 86 0F 0E                   06 01   PLAYER   DUCK
 49 2A 1C 3E 7F 7F 3E 5D 49 1C 1C 08 0C 02   PLAYER   TURTLE
 A2 66 F6 FE FE C6 BA FE D6 7C 38    0B 03   PLAYER   GHOST
 DECIMAL
 -->

SCR # 55
 ( RANDOM NUMBER WORDS )
 53770 CONSTANT RANDOM-REG           ( Atari random H/W register )
 : RAND                              ( mod --- random )
   RANDOM-REG C@                     ( Get pos 8bit random value )
   SWAP MOD ;                        ( Reduce to + 0,1,2,...mod-1 )
 : RANDOM                            ( mod --- random )
   RANDOM-REG C@ 256 *               ( Get MSB for random number )
   RANDOM-REG C@ +                   ( Get LSB, combine bytes )
   SWAP MOD ;                        ( Reduce to +/- 0,1,2,...mod-1 )
 : RANDOM-XY                         ( mod --- xinc yinc )
   DUP RANDOM                        ( Get +/-  random x-1 )
   SWAP RANDOM ;         -->         ( Get +/-  random y-1 )

SCR # 56
 ( MOVE PLAYER RANDOMLY )
 : RANDOM-MOVE                       ( pfa mod --- )
   RANDOM-XY                         ( Get random incx, incy )
   STEP-XY ;                         ( Step to new position )
 : RANDOM-TEST                       ( pfa mod --- )
```

(continued)

List 3 *(continued)*

```
  BEGIN                          ( Until a key is hit )
    2DUP                         ( Dup arguments )
    RANDOM-MOVE                  ( Move at random )
  ?TERMINAL UNTIL                ( Continue until a key )
  2DROP ;                        ( Clean up stack )

54016 CONSTANT PORTS             ( Base address for PORTA,B )
5     VARIABLE JOY-STEP          ( Increment for joy stick step )
-->


SCR # 57
( JOY STICK UTILITY WORD )

: JOY-XY                         ( stick --- xinc yinc )
  PORTS                          ( Base address for I/O ports )
  OVER 2 / +                     ( Add 1 if stick = 2,3 )
  C@                             ( Get the port value )
  SWAP 2 MOD IF                  ( Test for stick = 1,3 )
    16 /                         ( Yes, shift MSB down )
  ENDIF
  15 AND 15 XOR                  ( Clean, use positive logic )
  DUP 4 / DUP IF                 ( Test bits R,L for set )
    2 * 3 - JOY-STEP @ *         ( Yes, convert to +/- 1 )
  ENDIF                          ( Else leave zero for x )
  SWAP 3 AND DUP IF              ( Test D,U bits for set )
    2 * 3 - JOY-STEP @ *         ( Yes, convert to +/- 1 )
  ENDIF ;    -->                 ( Else leave zero for y )

SCR # 58
( MOVE PLAYER ACCORDING TO JOYSTICK )

: JOY-MOVE                       ( pfa stick --- )
  JOY-XY                         ( Get incx, incy from joystick )
  OVER ABS OVER ABS + IF         ( Joy stick set ? )
    STEP-XY                      ( Yes, step to new position )
  ELSE                           ( No, joy stick not in use )
    2DROP DROP                   ( Ignore )
  ENDIF ;

: JOY-TEST                       ( pfa stick --- )
  BEGIN                          ( Begin until key )
    2DUP                         ( Duplicate arguments )
    JOY-MOVE                     ( Move according to joy stick )
  ?TERMINAL UNTIL                ( Until a key is hit )
  2DROP ;               -->      ( Clean up stack )

SCR # 59
( DEFINE A HORIZONTAL MOVE )

: HORZ-MOVE                      ( pfa stepsize --- )
  OVER GET-XY                    ( Get the current xy position )
  ROT ROT +                      ( Add the stepsize to x )
  255 AND                        ( Keep in Player display space )
  SWAP                           ( Restore xy order on stack )
  MOVE-XY ;                      ( Move the Player )

-->

SCR # 60
( DEFINE A RANDOM JUMPING MOVE )

: JUMP-MOVE                      ( pfa +delx +dely --- )
  RANDOM-REG @ 3 AND 0= IF       ( Move 1 of 4 times )
    RANDOM                       ( Get random y: -dely<y<+dely )
    BYTES/PLAYER @ 2 /           ( Set the y center point )
    +                            ( Form new y position )
    SWAP RANDOM                  ( Get random x: -delx<x<+delx )
    128                          ( Set the x center point )
    +                            ( Form new x position )
    SWAP                         ( Restore xy order on stack )
    MOVE-XY                      ( Move Player )
  ELSE                           ( Do not move this time )
    DROP DROP DROP               ( Clean up stack )
  ENDIF ;
-->


SCR # 61
( INITIALIZE PLAYERS FOR DEMO )

: INIT-PLAYERS                   ( --- )
  4 8 1 SETCOLOR  2 8 1 SETCOLOR  1 8 1 SETCOLOR
  CLEAR-MEM
  COLLISION-OFF
  DUCK      0 BYTES/PLAYER @ 2 / SET-XY
  DUCK   15 10  PLYR-COLOR
  TURTLE 128 BYTES/PLAYER @ 2 / SET-XY
  TURTLE 12 4 PLYR-COLOR
  GHOST  128 BYTES/PLAYER @ 2 / SET-XY
  GHOST  0 15 PLYR-COLOR
  CROSS-HAIRS 128 20 SET-XY
  CROSS-HAIRS 4 8 PLYR-COLOR
  CROSS-HAIRS SHOW-XY ;
-->


SCR # 62
( DEMONSTRATE PLAYER MOTION )

: DEMO                           ( --- )
  46 P/M-SETUP                   ( Setup playfield, Players )
  BEGIN                          ( Loop until a key press )
    INIT-PLAYERS                 ( Initialize the Players )
    BEGIN                        ( Loop until a PO collision )
      DUCK          5 HORZ-MOVE  ( Move DUCK horizontally )
      TURTLE        3 RANDOM-MOVE ( The TURTLE wobbles around )
      GHOST   64 32 JUMP-MOVE    ( The GHOST jumps all around )
      CROSS-HAIRS 0 JOY-MOVE     ( Chase with the crosshairs )
    255 12 ?COLLISION UNTIL      ( Go until crosshairs gets one )
    BEEP                         ( Signal the hit )
  ?TERMINAL UNTIL KEY DROP       ( Repeat until key - ignore it )
  RESTORE-MEM ;                  ( Reset Atari screen/mem )
;S
```

Table 2: **Player Utilities Package Demonstration Glossary**

DEMO (—) .... Screen 62
Demonstrate the Player Utilities package with four types of motion and Players. The CROSS-HAIRS pattern is under joystick 1 control. To end DEMO, press any key other than BREAK and catch a player with the CROSS-HAIRS.

HORZ-MOVE (pfa stepsize —) .... Screen 59
Move the Player horizontally "stepsize" units (right is positive).

INIT-PLAYERS (—) .... Screen 61
Brute force initialization of players for the demonstration.

JOY-XY (stick — xinc yinc) .... Screen 57
Sample the joy "stick" (0-3) and convert to x,y displacements of +/− JOY-STEP units.

JOY-MOVE (pfa stick —) .... Screen 58
Move the player according to the joy "stick" position. If the joy stick is not being used then the player is not moved.

JOY-TEST (pfa stick —) .... Screen 58
Test the joy stick movement.

JUMP-MOVE (pfa +delx +dely —) .... Screen 60
Jump to a random position centered around the middle of the screen. The range of the random offset is +/− "delx" − 1 and +/− "dely"-1. This jump is performed, on the average, every fourth time JUMP-MOVE is executed. (The Atari RANDOM-REG is used to yield a 1 in 4 probability that JUMP-MOVE actually will be performed.)

RAND (mod — random) .... Screen 55
Get a positive random number between 0 and "mod"-1. "Mod" must be 255 or less.

RANDOM (mod — random) .... Screen 55
Get a random number in the range of +/− 0 ... "mod" − 1. "Mod" may be any 16-bit FORTH number.

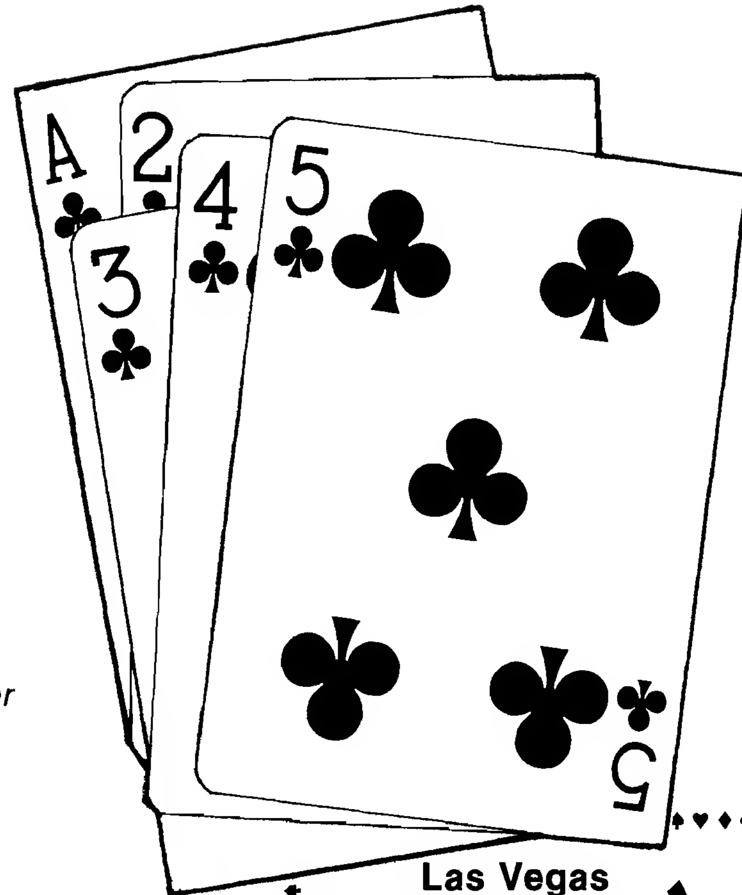
RANDOM-XY (mod — xinc yinc) .... Screen 55
Get random x,y increments in the range of +/−0 ... "mod" − 1.

RANDOM-MOVE (pfa mod —) .... Screen 56
Move the player at random from the current position. The step sizes are chosen randomly in the range of +/− 0 ... "mod"-1.

RANDOM-TEST (pfa mod —) .... Screen 56
Test the random move of a player.

MICRO

PASCAL SOLITAIRE



by Alan D. Floeter and Valerie A. Floeter

This Pascal program for the Apple allows the computer to play Las Vegas Solitaire.

The Solitaire program, though trivial as a game, points out some programming practices that work very well in Pascal, but can also be used in other languages. The main point is to break down the program into logical steps. Too many programming courses teach you to use subroutines only if you have some repetitive code, but subroutines make the program more readable and understandable. Pascal's procedures make it easy to get these results, but BASIC can use GOSUBs, and assembly language has JSRs. As a rough rule of thumb when we program — whether at home or work and in any language — we write code as subroutine calls, keeping each routine to roughly one page of paper. Keeping the functions separate makes debugging easier; you know where to look if a certain part of your program doesn't work. This is one area where a lot of new programmers lack skills.

The other technique used in this program deals with the type of data structure used. In this case Pascal really shines. Creating stacks for each pile of cards and then writing procedures to PUSH and POP not only makes the program look nicer, it makes it a lot easier to program. How you represent your data is very important.

Before starting the actual writing we broke the Solitaire game into several steps, with each step eventually becoming a procedure. For those unfamiliar with Pascal, you can think of a procedure as a subroutine. Pascal prefers procedures to be defined before they are used, so the actual mainline of the program is at the very end. This is sometimes bothersome because you have to go backwards through the program to understand what is going on.

The main procedures are SHUFFLE, DEAL, NEWCARD, PLAYOUT, MOVEWASTE, MOVEPILE, and DISPLAY.

The method used to SHUFFLE the deck of cards is very simple. A random

(Continued on page 111)

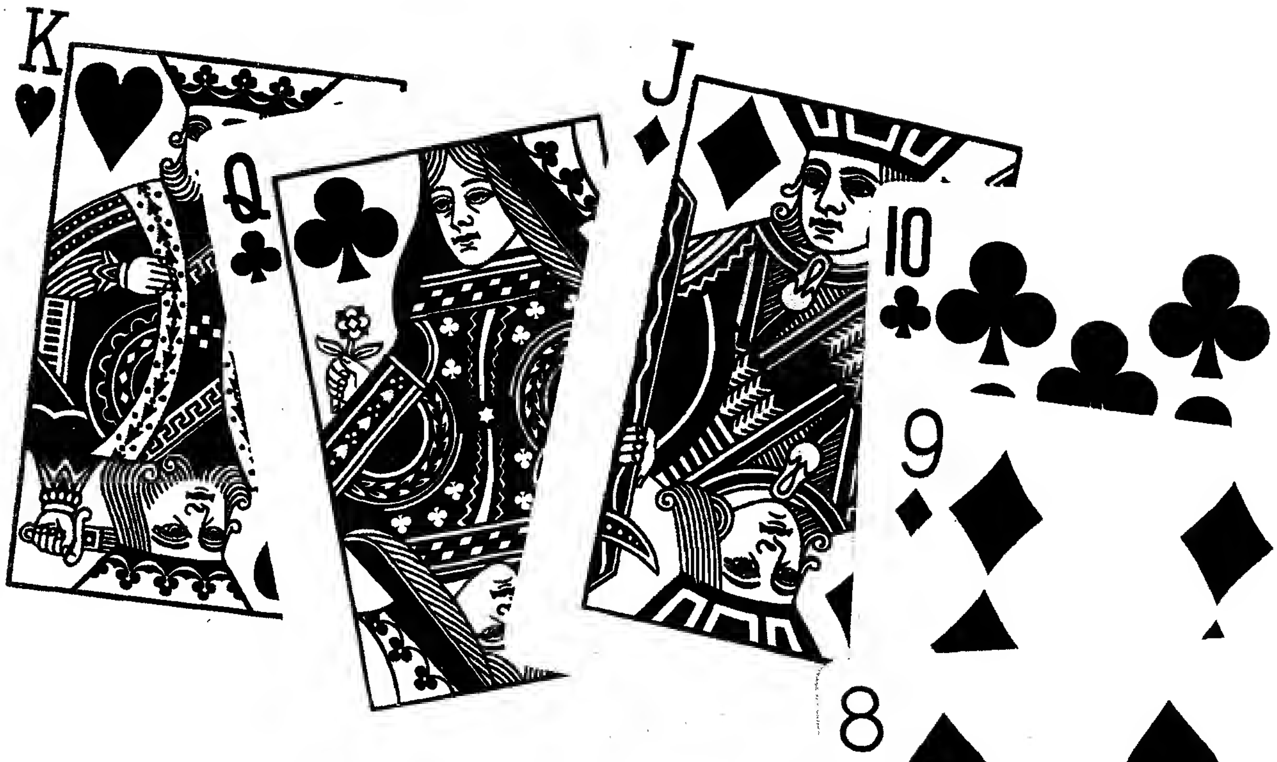
Solitaire requires:
Apple II with Pascal

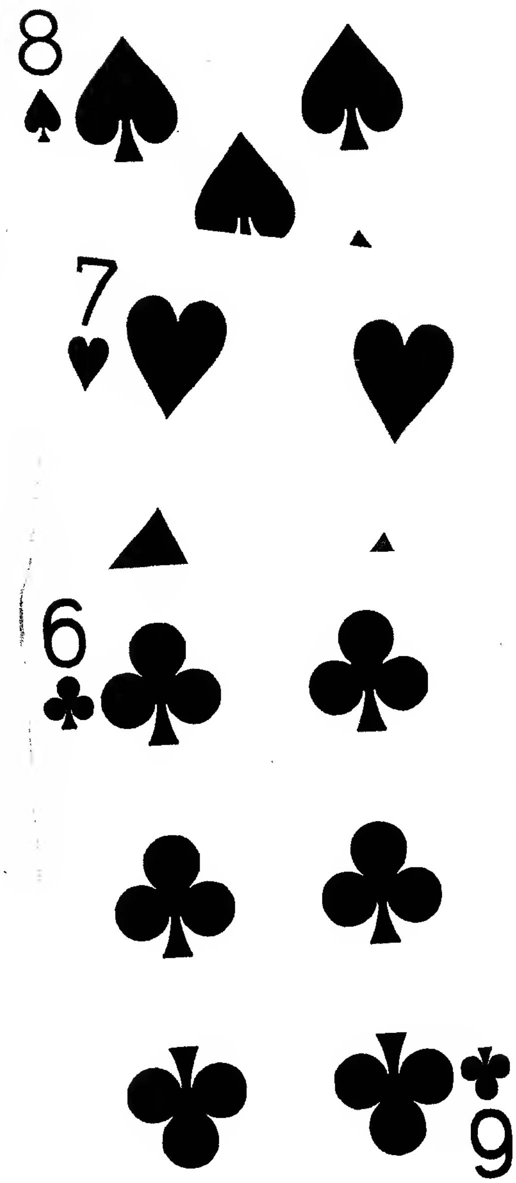
Las Vegas Solitaire Rules

In the book, *Fundamentals of Data Structures*, Ellis Horowitz and Sartaj Sahni explain the Las Vegas rules. Since they did such an excellent job, we will quote them here for those readers who are unfamiliar with the game.

"To begin the game, 28 cards are dealt into 7 piles. The leftmost pile has 1 card, the next has 2 cards, and so forth up to 7 cards in the rightmost pile. Only the uppermost card of each of the 7 piles is turned face up. The cards are dealt left to right, one card to each pile, dealing to one less pile each time, and turning the first card in each round face up.

"On the topmost face up card of each pile you may build in descending sequences red on black or black on red. For example, on the 9 of spades you may place either the 8 of diamonds or the 8 of hearts. All face up cards on a pile are moved as a unit and may be placed on another pile according to the bottommost face up card. For example, the 7 of clubs on the 8 of hearts may be moved as a unit onto the 9 of clubs or the 9 of spades.





"Whenever a face down card is uncovered, it is turned face up. If one pile is removed completely, a face-up King may be moved from a pile (together with all cards above it) or the top of the waste pile (see below) into the vacated space. There are four output piles, one for each suit, and the object of the game is to get as many cards as possible into the output piles. Each time an Ace appears at the top of a pile or the top of the stack it is moved into the appropriate output pile. Cards are added to the output piles in sequence, the suit for each pile being determined by the Ace on the bottom.

"From the rest of the deck, called the stock, cards are turned up one by one and placed face up on the waste pile. You may always play cards off the top of the waste pile, but only one at a time. Begin by moving a card from the stock to the top of the waste pile. If there is ever more than one possible play to be made, the following order must be observed:

"1. Move a card from the top of a playing pile or from the top of the waste pile to an output pile. If the waste pile becomes empty, move a card from the stock to the waste pile.

"2. Move a card from the top of the waste pile to the leftmost playing pile to which it can be moved. If the waste pile becomes empty move a card from the stock to the waste pile.

"3. Find the leftmost playing pile which can be moved and place it on top of the leftmost playing pile to which it can be moved.

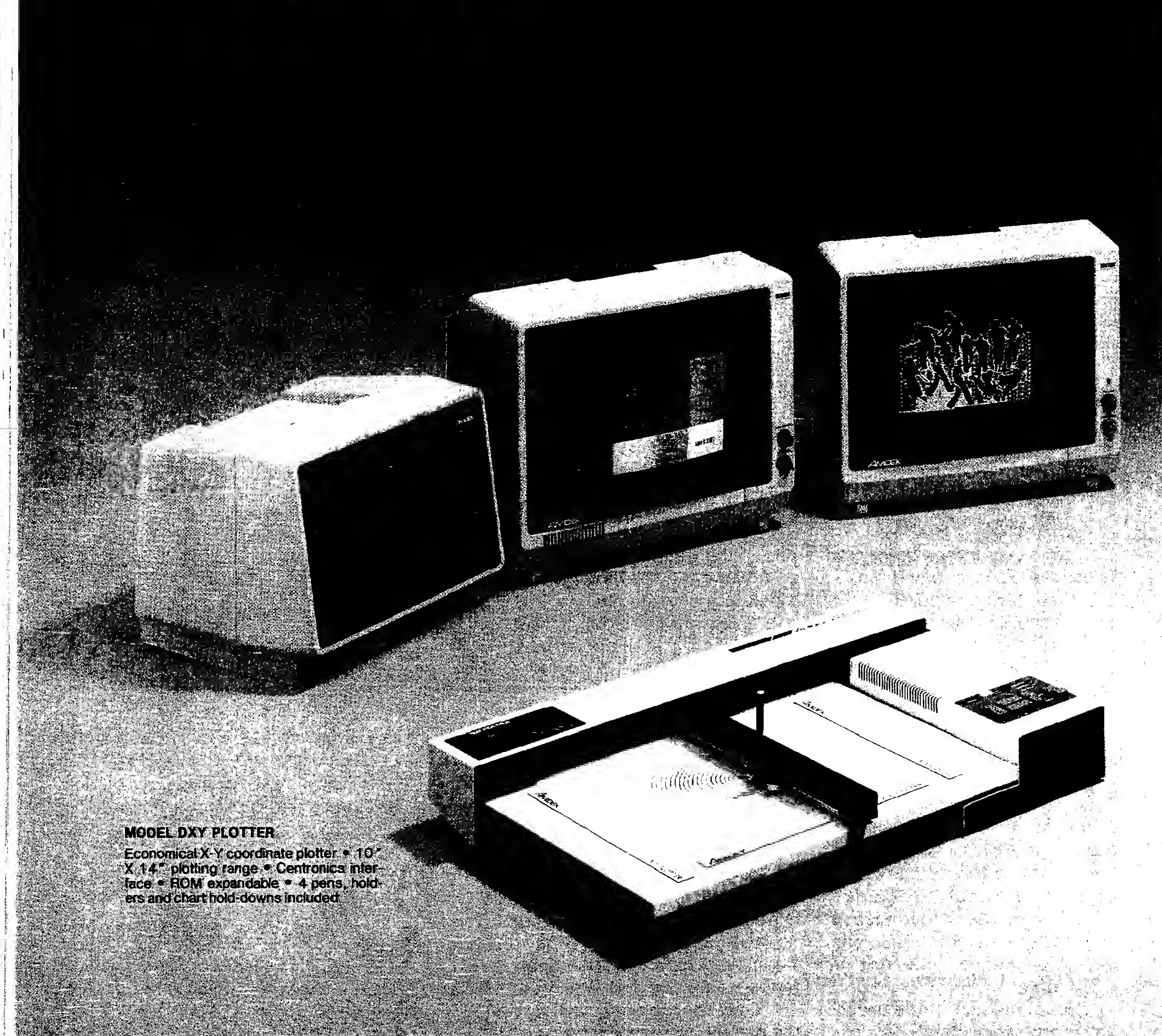
"4. Try 1, 2, 3 in sequence, restarting with 1 whenever a move is made.

"5. If no move is made *via* 1-4, move a card from the stock to the waste pile and retry 1.

"Only the topmost card of the playing piles or the waste pile may be played to an output pile. Once played on an output pile, a card may not be withdrawn to help elsewhere. The game is over when either all the cards have been played to the output or the stock pile has been exhausted and no more cards can be moved.

"When played for money, the player pays the house $52 at the beginning and wins $5 for every card played to the output piles."

AMDEK...your guide to



YOUR COMPATIBILITY CHART

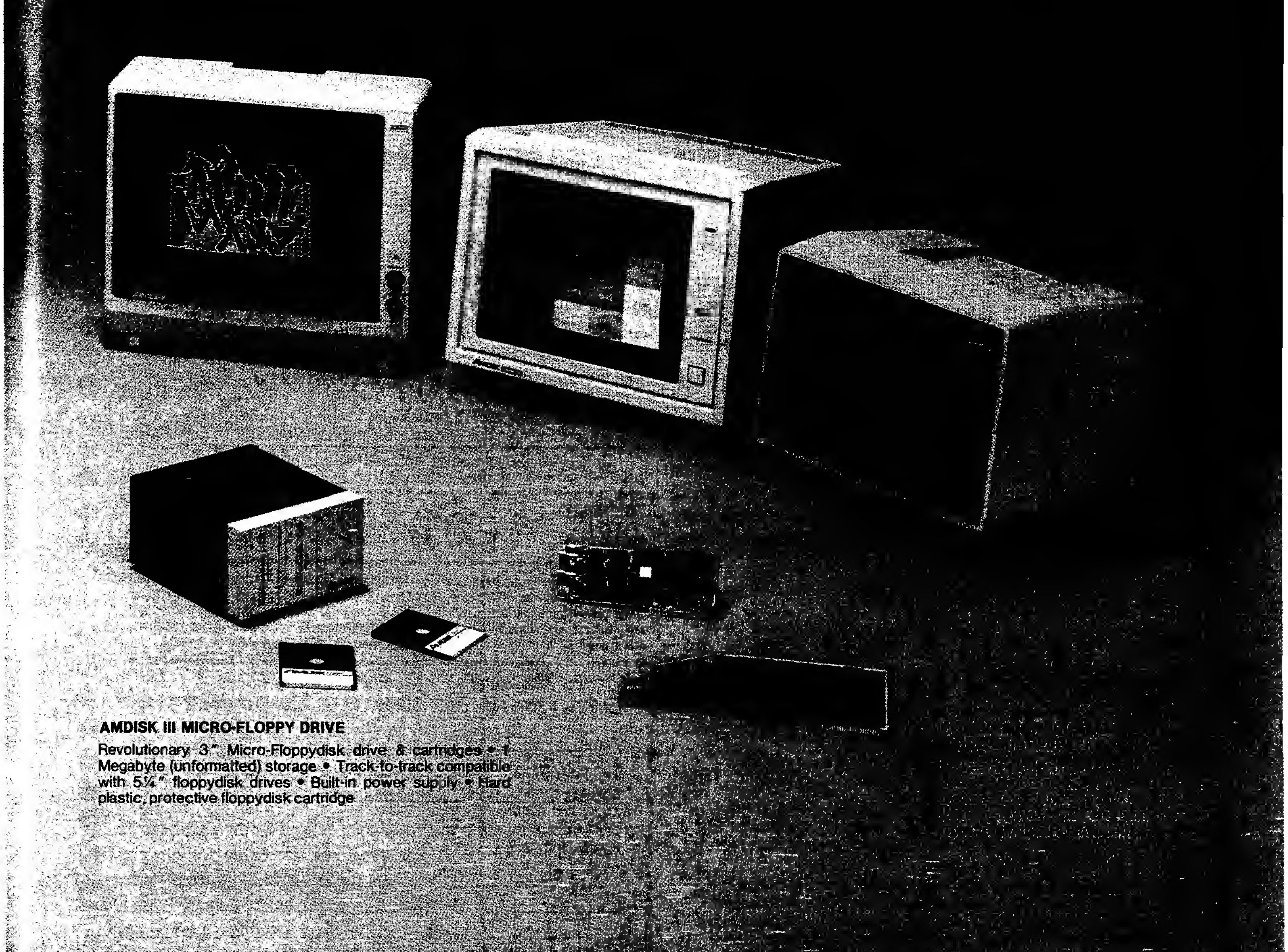
MORE COMPATIBILITY INTERFACES DUE SOON . . . CHECK FACTOR

COMPUTER	MONITORS							PLOTTER	AMDISK III	NOTES
	VIDEO-300	VIDEO-310	COLOR-I	COLOR-II	COLOR-IIA	COLOR-III	COLOR-IV			
IBM-PC	●	●	●	●	★	●		●	●	★ Special Cabling Required
APPLE III	●			★	★	★		●		★ Special Cabling or Converter Required
APPLE II	●		●	★	★	★		●		★ DVM Board Required
ATARI 800	★		★							★ Opt. Atari Cable Required
VIC-20			★							★ Opt. VIC Cable Required
TRS-80	★									★ Opt. TRS Cable Required
Osborne	★									★ Opt. Interface Required
TI-99			★							★ Opt. TI Cable Required
Commodore-64	★		★							★ Opt. Commodore Cable Req

innovative computing!

NEW 2 YEAR WARRANTY!

On all monitor electronics . 3 yrs. on all CRT's
(See details at dealer)



Amdek Corp. is dedicated to marketing quality computer peripheral equipment to enhance the use of popular personal computers. Our research & development staff keeps abreast of progress in computer techology and equipment and strives to offer you state-of-the-art advances in peripheral equipment.

Amdek products are distributed nationwide and in Canada through major distributors. And, we have factory-trained manufacturer's representatives ready to serve you in every major marketing area. Amdek offices are located in Chicago, Los Angeles & Dallas.

Just circle the reader service number, or contact us to receive complete technical specifications on these Amdek products.

AMDEK CORP.

2201 Lively Blvd. • Elk Grove Village, IL 60007
(312) 364-1180 TLX: 25-4786

REGIONAL OFFICES Calif (714) 662-3949 • Texas (817) 498-2334

Circle No. 67

number between 1 and 52 is chosen. If the card represented by the number has not been found yet, it is placed into the pile of shuffled cards. When all 52 cards are on the pile, they are dealt.

This shuffling method is easy to code, but it may take long to execute. When only a few cards are left to be placed in the pile, it may take many tries for their random numbers to appear. A much more elegant method would be to accept any 52 random numbers. These random numbers, each assigned to a card, are then sorted, which will shuffle their associated cards. Although we never noticed the shuffling process taking very long with the current method, the sorting improvement might be a nice enhancement to the program.

DEAL does as the name suggests; it takes a card from the deck and stores it in the appropriate pile. The NEWCARD routine takes a card from the stock pile and places it on the waste pile. PLAYOUT determines if any upturned cards can be played on the top four piles. MOVEWASTE checks to see if the card on the top of the waste pile can be played on any of the upturned piles. MOVEPILE tries to move any upturned piles onto other upturned piles. The DISPLAY routine displays the top card of a pile on the screen. Each card is displayed on the screen as a two-character code. An "XX" means the card is turned face down. Otherwise, the first character is the number — A for ace, 2-9 for their respective numbers, T for ten, J for jack, Q for queen, and K for king. The second character is the suit — D diamonds, H hearts, C Clubs, and S spades. Therefore, a TH on the screen represents the ten of hearts.

To handle all the piles of cards, we used arrays to form stacks. There are 19 stacks of cards — 4 piles for each suit on top, 6 downturned piles, 7 upturned cards to be played on, a waste pile, and the stock pile. Rather than having a separate array for each, a two-dimensional array called STACK is used. The piles are numbered as follows:

```
          16    17    18    19
     10    11    12    13    14    15
1     2     3     4     5     6     7
                8     9
```

The pile number represents the first dimension of the array, so STACK[8,X] indicates a card in pile 8. Since each pile of cards is treated as an individual stack, the PUSH and POP routines will put a card on the stack or take one off, respectively.

Other arrays used in conjunction with the stack include the array stack. This keeps track of where the next card should go on each stack, like a stack pointer for each stack. The arrays XP and YP store the address on the screen where each pile should be displayed. FIRST and SECOND contain the characters to display for each card. DECK is the shuffled deck of cards. Knowing about these arrays should help you understand the Solitaire program.

You enter the Solitaire program from the Pascal editor and compile. It uses the APPLESTUFF file provided on the master disk. (This is to get the RANDOM function.) Our version was compiled on Pascal 1.0, but version 1.1 should work as well.

When you execute (or run) the program it will ask you how many games you care to play. Once you enter your choice, the cards are shuffled, dealt, and play begins. You can stop play by hitting any key. The game will continue when you press a key.

After a game is finished, a short pause — about four seconds — allows the spectators to see the results before starting the next game. When the number of games requested has been played, the final winnings are scored.

Due to time constraints on our part, the display routine is not complicated. The next step in the development of this program would be to come up with a graphics display. The only procedure affected is DISPLAY, thanks to our structure approach, so feel free to experiment.

Valerie and Alan Floeter have formed a consulting firm known as The Software Experience, which supports businesses owning Apples. They are the authors of THE ASSEMBLER and MacroSoft and have published about 15 articles in computer magazines. Also, they have designed some hardware boards for the Apple, which are marketed by other firms. Al is a senior systems engineer for Wisconsin Electrical Manufacturing in New Berlin, WI. You can contact the Floeters at 4333 N. 71 St., Milwaukee, WI 53216.

Program Solitaire

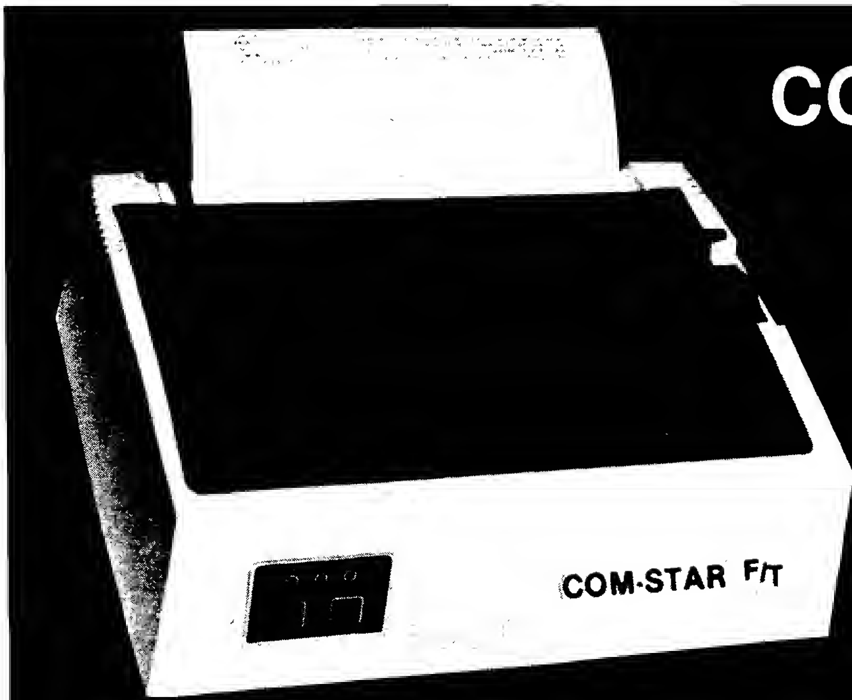
```
{ This program plays the card game of Solitaire.  The cards are displayed
  on the screen as play is occuring.  The user enters the number of games to
  play.  The program outputs net winnings based on $52 payed to the
  house for each game and $5 won back for each card output.

  There are 19 stacks in use:

                 - PLAY CARDS -
                 16  17  18  19
             ---- DOWN CARDS ----
             10  11  12  13  14  15
           ------ UP CARDS ------
           1   2   3   4   5   6   7
              --- WASTE CARDS ---
                     8   9

         Numbers  1-13 = AC - KC
                 14-26 = AS - KS
                 27-39 = AD - KD
                 40-52 = AH - KH

            (See Page 104, #27). }

USES APPLESTUFF;

CONST NEW=1;            { Constants for later use }
      FILWASTE=2;
      PLAYO=3;
      MOVEW=4;
      MOVEP=5;
      PILE=9;
      WAST=8;

VAR   DECK: ARRAY[1..52] OF INTEGER;
      STACK: ARRAY[1..19,1..52] OF INTEGER;
      POINT,XP,YP: ARRAY[1..19] OF INTEGER;
      FIRST: ARRAY[1..4] OF STRING;
      SECOND: ARRAY[1..13] OF STRING;
      GOOD2,GOOD,CHANGE,QUIT,EMPTY: BOOLEAN;
      TEMP,WHICH,MOVE,UP,DN,CARD,CARDOUT,UN,II,TR,
      INP,DIS,SUIT,NUM,X,Y,NUMB1,NUMB2,COL1,COL2: INTEGER;
      NUMGAM,GAME,WINNING:INTEGER;
      CHCT: STRING;
      FCH,SCH: STRING;
      CH:CHAR;
```

(continued)

Program Solitaire *(continued)*

```
PROCEDURE INIT;          { Initialize variables }
  BEGIN
      FOR UN:=1 TO 52 DO DECK[UN]:=0;
      FOR UP:=1 TO 19 DO POINT[UP]:=0; { Stack empty }

      FIRST[1]:='C';
      FIRST[2]:='S';
      FIRST[3]:='D';
      FIRST[4]:='H';

      SECOND[1]:='A';
      SECOND[2]:='2';
      SECOND[3]:='3';
      SECOND[4]:='4';
      SECOND[5]:='5';
      SECOND[6]:='6';
      SECOND[7]:='7';
      SECOND[8]:='8';
      SECOND[9]:='9';
      SECOND[10]:='T';
      SECOND[11]:='J';
      SECOND[12]:='Q';
      SECOND[13]:='K';

      { The following are X and Y positions on the screen for each stack }
      FOR UP:=1 TO 7
          DO YP[UP]:=5;

      YP[8]:=22;
      YP[9]:=22;

      FOR UP:=10 TO 15 DO YP[UP]:=4;

      YP[16]:=2;
      YP[17]:=2;
      YP[18]:=2;
      YP[19]:=2;

      XP[1 ]:=10;
      XP[2 ]:=13;
      XP[ 3]:=16;
      XP[ 4]:=19;
      XP[ 5]:=22;
      XP[ 6]:=25;
      XP[ 7]:=28;
      XP[ 8]:=16;
      XP[ 9]:=22;
      XP[10]:=13;
      XP[11]:=16;
      XP[12]:=19;
      XP[13]:=22;
      XP[14]:=25;
      XP[15]:=28;
      XP[16]:=16;
      XP[17]:=19;
      XP[18]:=22;
      XP[19]:=25;

      END;

PROCEDURE DISPLAY(I:INTEGER;PUT:BOOLEAN);
{ Input stack number and false for back of card, true for front }
  BEGIN
    IF ((I>8) AND (I<16)) THEN CHCT:='XX' { If dwn pile, dsply back }
         ELSE BEGIN
             DIS:=STACK[I,POINT[I]];       { Get card }
             SUIT:=TRUNC((DIS-1) / 13) +1; { Get suit 1-4 }
             NUM:=((DIS-1) MOD 13+1);      { Get # 1-13 }
             FCH:=FIRST[SUIT];
             SCH:=SECOND[NUM];
             CHCT:= CONCAT(SCH,FCH)
         END;
      IF PUT=FALSE                         { If erase }
         THEN IF CHCT='XX'                 { Then check if display back }
                THEN BEGIN
                    IF POINT[I]=1          { If none left, then blank it }
                       THEN CHCT:='  '
                END
         ELSE CHCT:='  ';
      X:=XP[I];                            { Get x pos }
      Y:=YP[I];                            { And y pos }
      IF I<8 THEN Y:=Y+POINT[I]-1;       { If up pile, can go down more }
      GOTOXY(X,Y);                         { Do it }
      WRITELN(CHCT)                        { Write it out }
  END;

PROCEDURE PUSH(I:INTEGER;A:INTEGER);
{ Input stack number and item }
  BEGIN
      IF POINT[I]>=52
          THEN BEGIN
              WRITELN('STACK OVERFLOW');
              HALT
              END;
      POINT[I]:=POINT[I]+1;                { Update stack pointer }
      STACK[I,POINT[I]]:=A;                { Put value in }
      DISPLAY(I,TRUE)                      { And display it }
  END;

PROCEDURE POP(I:INTEGER;VAR A:INTEGER);
{ Input stack number and get item back }
   BEGIN
       IF POINT[I]<=0
           THEN BEGIN
               WRITELN('STACK UNDERFLOW');
               HALT
           END;
       A:=STACK[I,POINT[I]];               { Get from stack }
       DISPLAY(I,FALSE);                   { And erase it }
       POINT[I]:=POINT[I]-1                { Decrement pointer }
   END;

PROCEDURE LOOK(I:INTEGER;VAR A:INTEGER;VAR GOOD:BOOLEAN);
{ Input stack number and get top item back and true if not empty, false
  if empty }
   BEGIN
       IF POINT[I]=0                       { Stack empty? }
           THEN BEGIN
               GOOD:=FALSE;                { Yes, say so }
               A:=0                        { Default }
           END
           ELSE BEGIN
               GOOD:=TRUE;                 { Stack has stuff }
               A:=STACK[I,POINT[I]]        { Get it }
           END
   END;

PROCEDURE SHUFFLE;
{ Snuffle cards }
   BEGIN
       FOR INP:=1 TO 52
           DO BEGIN
              REPEAT
                  CARD:=RANDOM MOD 52 +1;  { Get number from 1 to 52 }
                  UNTIL DECK[CARD]=0;      { See if this is used }
              DECK[CARD]:=1;               { Now it is used }
              PUSH(PILE,CARD)              { Push it on the pile }
           END;
   END;

PROCEDURE DEAL;
{ Deal cards }
   BEGIN
       FOR UP:=1 TO 7 DO
           BEGIN
               POP(PILE,CARD);             { Get card }
               PUSH(UP,CARD);              { Put on up pile }
               FOR DN:=UP+9 TO 15 DO       { Rest of down piles }
                   BEGIN
                       POP(PILE,CARD);     { Get card }
                       PUSH(DN,CARD)       { Put on down pile }
                   END
           END;
   END;

PROCEDURE NEWCARD;
{ Get new card to waste pile }
  BEGIN
      LOOK(PILE,CARD,GOOD);                { Any left? }
      IF GOOD=FALSE                        { No }
          THEN EMPTY:=TRUE                 { Say its empty }
          ELSE BEGIN
              EMPTY:=FALSE;
              POP(PILE,CARD);              { Get card }
              PUSH(WAST,CARD)              { And put on waste pile }
          END;
      MOVE:=PLAYO;                         { Next play }
  END;
```

(Continued on page 114)



COM-STAR F/T

Tractor Friction Printer

only $279

- Lowest price quality tractor friction printer in the U.S.A. • Fast 80 characters per second
- 40, 46, 66, 80, 96, or 132 characters per line spacing • Prints labels, letters, graphs, and tables
- List your programs • Print out data from modem services

Deluxe COMSTAR F/T PRINTER — $279.00

The Comstar is an excellent addition to any micro-computer system. (Interfaces are available for Apple, VIC-20, Commodore-64, Pet, Atari 400 and 800, and Hewlett Packard) At only $279. the Comstar gives you print quality and features found only on printers costing twice as much. Compare these features.

- **BI-DIRECTIONAL PRINTING** with a LOGIC SEEKING CARRIAGE CONTROL for higher through-put in actual text printing. 80 characters per second.
- **PRINTING VERSATILITY:** standard 96 ASCII character set plus block graphics and international scripts. An EPROM character generator includes up to 224 characters.
- **INTERFACE FLEXIBILITY:** Centronics is standard. Options include EIA RS232C, 20mA Current Loop. (Add $20.00 for RS232)
- **LONG LIFE PRINT HEAD:** 100 million character life expectancy.
- **THREE SELECTABLE CHARACTER PITCHES:** • 10, 12 or 16.5 characters per inch. 132 columns maximum. Double-width font also is standard for each character pitch.
- **THREE SELECTABLE LINE SPACINGS:** 6, 8 or 12 lines per inch.
- **PROGRAMMABLE LINE FEED:** programmable length from 1/144 to 255/144 inches.
- **VERTICAL FORMAT CONTROL:** programmable form length up to 127 lines, useful for short or over-sized preprinted forms.
- **FRICTION AND TRACTOR FEED:** will accept single sheet paper.
- **224 TOTAL CHARACTERS**
- **USES STANDARD SIZE PAPER**

If you want more try —

Premium Quality COMSTAR F/T SUPER-10" PRINTER — $329.00

More Features Than MX-80 For $250 Less

For $329.00 you get all of the features of the Comstar plus 10" carriage, 100 cps, 9 x 9 dot matrix with double strike capability for 18 x 18 dotmatrix. High resolution bit image (120 x 144 dot matrix), underlining, backspacing, 2.3K buffer, left and right margin settings, true lower descenders, with super and subscripts, and prints standard, italic, Block Graphics, special characters, plus 2K of user definable characters. For the ultimate in price performance the Comstar F/T Super 10" leads the pack!

80 COLUMN PRINTER $199

Super silent operation, 60 CPS, prints Hi-resolution graphics and block graphics, expanded character set, exceptionally clear characters, fantastic print quality, uses inexpensive thermal roll paper!

Double Immediate Replacement Warranty

We have doubled the normal 90 day warranty to 180 days. Therefore if your printer fails within "180 days" from the date of purchase you simply send your printer to us via United Parcel Service, prepaid. We will IMMEDIATELY send you a replacement printer at no charge via United Parcel Service, prepaid. This warranty, once again, proves that WE LOVE OUR CUSTOMERS!

15 DAY FREE TRIAL

OTHER OPTIONS

Extra Ribbons	$ 5.95
Roll Paper Holder	32.95
Roll Paper	4.95
5000 Labels	19.95
1100 Sheets Fan Fold Paper	13.95

Add $20.00 shipping, handling and insurance. Illinois residents please add 6% tax. Add $40.00 for CANADA, PUERTO RICO, HAWAII, ALASKA orders. WE DO NOT EXPORT TO OTHER COUNTRIES. Enclose cashiers check, money order or personal check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail available!! Canada orders must be in U.S. dollars.

PROTECTO ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

COMSTAR F/T

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijk
lmn opqrstuvwxyz 1234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890

SUPER-10" ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890

Circle No. 69

Program Solitaire *(continued)*

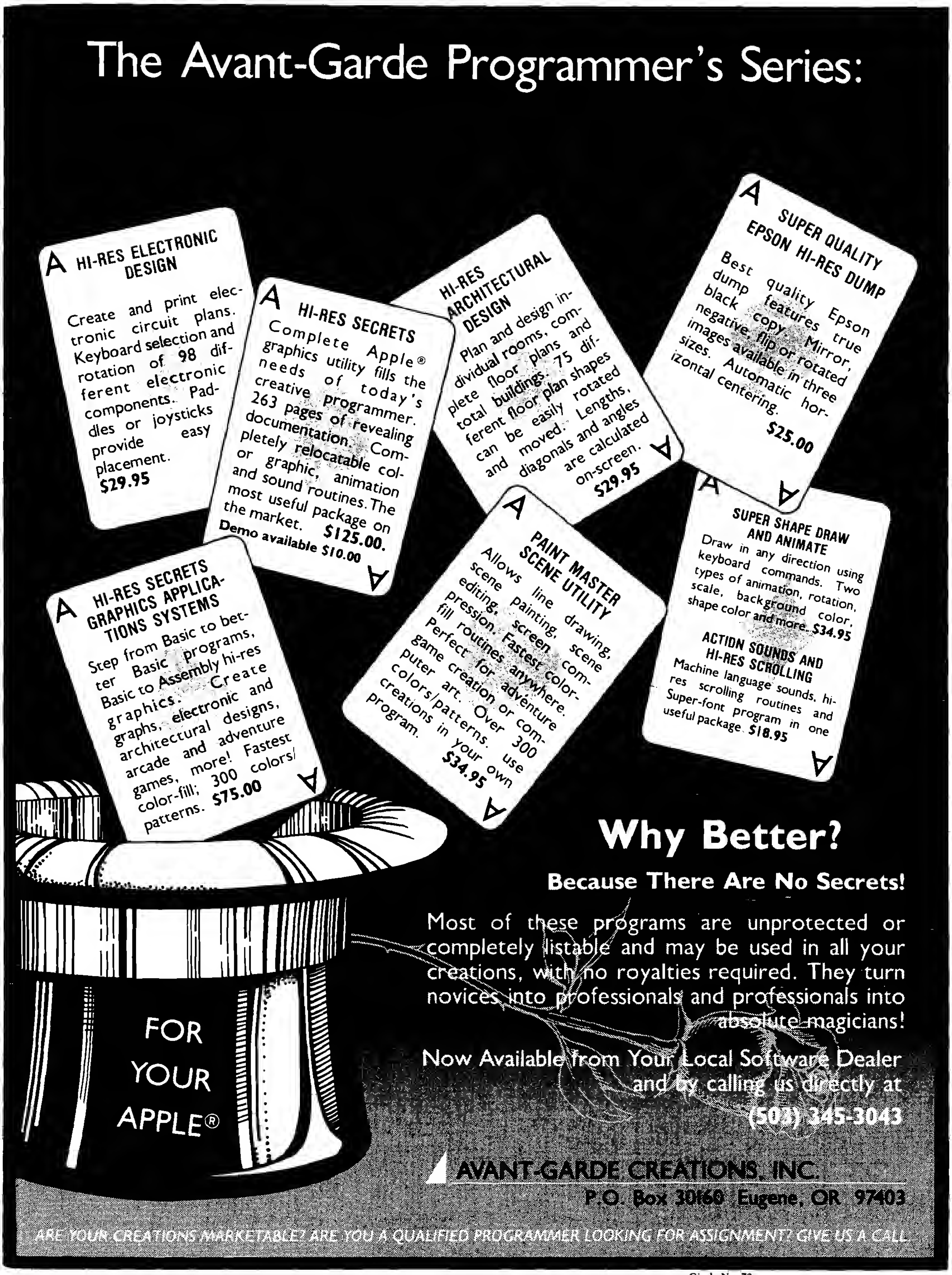
```
PROCEDURE TURN(I:INTECER);
{ Input stack number to turn up }
   BECIN
      IF ((I>1) AND (I<8))        { Only turn up cards for 2-7 piles }
         THEN BECIN
            II:=I+8;                   { Get down pile associated with it }
            LOOK(II,TEMP,GOOD2);       { One there? }
            IF GOOD2=TRUE              { Yes }
               THEN BEGIN
                  POP(II,TEMP);        { Get it }
                  PUSH(I,TEMP)         { Display it }
               END
         END
   END;

PROCEDURE PLAYOUT;
{ Play to output piles if any }
   BECIN
      MOVE:=MOVEW;                       { Do this next if no move }
      FOR UP:=1 TO 8
         DO BECIN
            LOOK(UP,CARD,GOOD);          { Card on up pile? }
            IF GOOD=TRUE                 { If yes }
               THEN BEGIN
                  WHICH:=TRUNC((CARD-1)/13) +16;  { Get which suit }
                  LOOK(WHICH,CARDOUT,GOOD);       { Out already }
                  IF GOOD=FALSE
                     THEN CARDOUT:=(WHICH-16)*13;
                  IF CARD=CARDOUT+1
                     THEN BEGIN
                        POP(UP,CARD);               { Take card off }
                        PUSH(WHICH,CARD);           { Play out }
                        WINNING:=WINNING+5;
                        LOOK(UP,CARD,GOOD);         { If up pile empty }
                        IF GOOD=FALSE THEN TURN(UP); { Turn up a card }
                        MOVE:=FILWASTE;    { Maybe replace waste pile }
                        IF UP=WAST THEN
                           BEGIN
                           LOOK(UP,CARD,GOOD);
                           { Display card underneath }
                           IF GOOD =TRUE THEN DISPLAY(WAST,TRUE);
                           END;
                     END
               END
         END;
   END;

PROCEDURE FILLIT;
{ Keep waste pile not empty }
   BEGIN
      MOVE:=PLAYO;
      LOOK(WAST,CARD,COOD);                { Any on waste pile? }
      IF COOD=FALSE THEN MOVE:=NEW         { If no, then get a new card }
   END;

PROCEDURE CHECK(VAR CHANCE:BOOLEAN);
{ Card and Cardout are the two cards, change comes back, true if play
   available, false if not }
   BEGIN
      CHANGE:=FALSE;
      NUMB1:=((CARD-1) MOD 13+1);          { Cet # }
      NUMB2:=((CARDOUT-1) MOD 13+1);
      IF (CARDOUT=Ø) AND (NUMB1=13)        { If empty and a king }
         THEN CHANGE:=TRUE;
      IF NUMB1+1=NUMB2                     { If numeric order correct }
         THEN BEGIN
            COL1:=TRUNC(CARD/27);          { Cet color }
            COL2:=TRUNC(CARDOUT/27);       { Of each }
            IF COL1<>COL2                   { If color different }
               THEN CHANGE:=TRUE
         END
   END;

PROCEDURE MOVEWASTE;
{ See if waste card can be played on up cards }
   BEGIN
      MOVE:=MOVEP;                         { This next if no move }
      LOOK(WAST,CARD,GOOD);                { Look at waste pile }
      IF GOOD=TRUE                         { If not empty }
         THEN BEGIN
            FOR UP:=1 TO 7
               DO BEGIN
                  LOOK(UP,CARDOUT,GOOD);   { See if card is up }
                  CHECK(CHANGE);           { Is there a play? }
                  IF CHANGE=TRUE           { If yes }
                     THEN BEGIN
                        POP(WAST,CARD);    { Get card }
                        PUSH(UP,CARD);     { And put on up pile }
                        MOVE:=FILWASTE;    { Maybe get new
                                             waste card }
                        UP:=7;             { And get out }
                        LOOK(WAST,CARD,GOOD);
                        IF GOOD =TRUE THEN
                          DISPLAY(8,TRUE); { Display it again }
                     END
               END
         END
   END;

PROCEDURE TRANF(FROMC:INTEGER;TOC:INTEGER);
{ Transfer up cards FROMC to up cards TOC }
VAR TEMP2:INTEGER;
    BEGIN
       TR:=POINT[FROMC];           { Depth of stack }
       FOR TEMP2:=1 TO TR
          DO BEGIN
             POINT[FROMC]:=TEMP2;
             TEMP:=STACK[FROMC,POINT[FROMC]];        { Get under }
```

(Continued on page 116)

Program Solitaire *(continued)*

```
                DISPLAY(FROMC,FALSE);                    { Erase it }
                PUSH(TOC,TEMP);                          { Put on other }
                DISPLAY(TOC,TRUE)                        { Display it }
              END;
            POINT[FROMC]:=Ø                              { Empty stack }
          END;

    PROCEDURE MOVEPILE;
    { See if pile should be moved to another pile }
      BEGIN
        MOVE:=NEW;
         FOR UN:=1 TO 7
           DO BEGIN
              LOOK(UN,CARD,GOOD);     { Is there a card up? }
              IF GOOD=TRUE
                THEN BEGIN
                  FOR UP:=1 TO 7
                    DO BEGIN
                      LOOK(UP,CARDOUT,GOOD);  { Get top card }
                      CARD:=STACK[UN,1];      { Get under card }
                      CHECK(CHANGE);          { Playable? }
                      IF CHANGE=TRUE          { If yes }
                        THEN BEGIN
                          LOOK(UN+8,TEMP,GOOD);
                          IF((NUMB1<>13)OR((GOOD=TRUE)AND(UN<>1)))
                          THEN BEGIN
                            TRANF(UN,UP); { Move pile }
                            TURN(UN);     { Turn up new }
                            UN:=7;        { Quit }
                            UP:=7;
                            MOVE:=PLAYO
                          END
                        END
                    END
                END
            END;

        IF ((MOVE=NEW)AND(EMPTY=TRUE))THEN QUIT:=TRUE  { Stop }
      END;

    PROCEDURE CHECKKEY;
    { Pause for key hit }
      BEGIN
        IF KEYPRESS=TRUE THEN     { If key hit }
          BEGIN
            READ(CH);
            REPEAT CH:=CH;        { Pause until key hit }
            UNTIL KEYPRESS;
            READ(CH);
          END;
      END;

    BEGIN
        RANDOMIZE;
        GAME:=1;
        WINNING:=Ø;
        WRITELN('ENTER NUMBER OF GAMES TO PLAY');
        READLN(NUMGAM);
        WHILE GAME<=NUMGAM DO
          BEGIN
            GOTOXY(Ø,23);FOR II:=1 TO 24 DO WRITELN;  { Clear screen }
            WRITELN('GAME NUMBER ',GAME);
            WINNING:=WINNING-52;
            INIT;
            SHUFFLE;
            DEAL;
            QUIT:=FALSE;
            MOVE:=NEW;
            REPEAT
            BEGIN
              CHECKKEY;                          { Pause if key hit }
              CASE MOVE OF
                NEW: NEWCARD;
                FILWASTE: FILLIT;
                PLAYO: PLAYOUT;
                MOVEW: MOVEWASTE;
                MOVEP: MOVEPILE
              END;
            END;
            UNTIL QUIT=TRUE;
            FOR TEMP:=1 TO 10000 DO TEMP:=TEMP; { Pause a bit }
            GAME:= GAME+1;
          END;

        GOTOXY(Ø,23);
        WRITELN('NET WINNING ',WINNING)
    END.
```

MICRO



The Avant-Garde Programmer's Series:
A HI-RES ELECTRONIC DESIGN
Create and print electronic circuit plans. Keyboard selection and rotation of 98 different electronic components. Paddles or joysticks provide easy placement.
$29.95
A HI-RES SECRETS
Complete Apple® graphics utility fills the needs of today's creative programmer. 263 pages of revealing documentation. Completely relocatable color graphic, animation and sound routines. The most useful package on the market. $125.00.
Demo available $10.00
HI-RES ARCHITECTURAL DESIGN
Plan and design individual rooms, complete floor plans and total buildings. 75 different floor plan shapes can be easily rotated and moved. Lengths, diagonals and angles are calculated on-screen.
$29.95
A SUPER QUALITY EPSON HI-RES DUMP
Best quality Epson dump features true black copy. Mirror, negative, flip or rotated images available in three sizes. Automatic horizontal centering.
$25.00
A HI-RES SECRETS GRAPHICS APPLICATIONS SYSTEMS
Step from Basic to better Basic programs, Basic to Assembly hi-res graphics. Create graphs, electronic and architectural designs, arcade and adventure games, more! Fastest color-fill; 300 colors/patterns. $75.00
A PAINT MASTER SCENE UTILITY
Allows line drawing, scene painting, scene editing, screen compression. Fastest color-fill routines anywhere. Perfect for adventure game creation or computer art. Over 300 colors/patterns. Use creations in your own program.
$34.95
A SUPER SHAPE DRAW AND ANIMATE
Draw in any direction using keyboard commands. Two types of animation, rotation, scale, background color, shape color and more. $34.95
ACTION SOUNDS AND HI-RES SCROLLING
Machine language sounds, hi-res scrolling routines and Super-font program in one useful package. $18.95
FOR YOUR APPLE®
Why Better?
Because There Are No Secrets!
Most of these programs are unprotected or completely listable and may be used in all your creations, with no royalties required. They turn novices into professionals and professionals into absolute magicians!
Now Available from Your Local Software Dealer and by calling us directly at
(503) 345-3043
AVANT-GARDE CREATIONS, INC.
P.O. Box 30160 Eugene, OR 97403
ARE YOUR CREATIONS MARKETABLE? ARE YOU A QUALIFIED PROGRAMMER LOOKING FOR ASSIGNMENT? GIVE US A CALL.

Circle No. 72

more 68000 addressing modes

by Joe Hootman

Our last installment in the 68000 series

Address Register Indirect with Index and Displacement

The address register indirect with index and displacement is similar to the address register indirect with displacement except that a data or address register can add to the displacement. The displacement is a sign-extended integer specified in the post word of the opcode. The effective address is calculated by adding the specified address register contents to the designated data or address register contents and adding the sign-extended 8-bit displacement integer to the results.

The extension word for the address register indirect with index and displacement is given below:

15	14 13 12	11	10	9	8	7 6 5 4 3 2 1 0
D/A	Register	W/L	0	0	0	Displacement

D/A — Index register displacement
- 0 — data register
- 1 — address register

Register — Number of index register

W/L
- 0 — sign-extended low order word in index register
- 1 — Uses the entire word in the index register

The next example illustrates this addressing mode. Suppose the data register D0 is to be loaded with the data in the memory location defined by A0, D1 and an offset of −4. The data register D1 is considered to be the "Index" register. The coding of the desired instruction follows:

```
MOVE.W-4(A0,D1.W),D0 { 3030 } Opword
                     { 10FC } Extension
                              word

Before execution    After execution
D0  00000000        D0  00000405
D1  00000008        D1  00000008
A0  00001500        A0  00001500
```

Memory

Address	Data
15FB	FB
15FC	FC
15FD	FD
15FE	FE
15FF	FF
1500	0F
1501	01
1502	02
1503	03
1504	04
1505	05
1506	06

The address register indirect with index and displacement is very powerful. This particular addressing mode is useful when a series of tables are stored in memory. The address register can be loaded with the beginning address of the series of tables. The index register can then select the particular table and the displacement will allow the selection of the particular data in the table.

Absolute Short Address

When using the absolute short address the effective address of the data is included in the instruction. If you want to move data to D0 from memory locations \$1500 and \$1501 using the absolute short addressing mode, then the mnemonic for the operation is

```
MOVE.W $1500,D0 { 3038 Opword
                { 1500 Extension word
```

This instruction fetches the data from memory locations \$1500 and \$1501 and places the data in bits 15-0 of D0.

Absolute Long Address

The absolute long addressing mode is similar to the short addressing mode except it allows access to all the memory space. If D0 is to be loaded using the long addressing mode,

```
MOVE.W $11500,D0 { 3039   Opword
                 { 0001 }
                 { 1500 } Extension Address
```

Note that the absolute long addressing mode uses two extension words while the absolute short addressing mode uses only one extension word.

Program Counter With Displacement/Indexed Addressing Mode

The program counter addressing mode is used when programs are to be independent of the location of the program in memory. This allows the programmer to code the program into a ROM and to be able to use the ROM in any 68000 microprocessor and at any memory location in the system.

There are two addressing methods used in the program counter addressing mode: the program counter with displacement and program counter with index. Both of the addressing modes are based on the same principle, which is that the program counter points to the location of the program and offsets are calculated from the program counter position.

Program Counter with Displacement

The program counter with displacement forms the effective address by adding a fixed offset to the present value of the PC. The fixed offset is a 16-bit sign-extended integer. The value of the PC is the address of the extension word. The data at the effective address is retrieved from the effective address. The displacement is contained in an extension word. To load D0 from an offset of + 8 from the PC the following opcode could be used.

(Continued)

Table 1: **Addressing Modes**

Name of Addressing Mode	Effective Address Modes	Mode	Register	Syntax
1. Data Register Direct	Dn	000	register no.	Dn
2. Address Register Direct	An	001	register no.	An
3. Address Register Indirect	An@	010	register no.	(An)
4. Address Register Indirect with Postincrement	An@ +	111	register no.	(An) +
5. Address Register Indirect Predecrement	An@ −	100	register no.	−(An)
6. Address Register Indirect with Displacement	An@(d)	101	register no.	d(An)
7. Address Register Indirect with Index	An@(d,ix)	110	register no.	d(An,Ri)
8. Absolute Short Address	XXX.W	111	000	xxx
9. Absolute Long Address	XXX.L	111	001	xxxxxx
10. Program Counter with Displacement	PC@(d)	111	010	PC relative +d16
11. Program Counter with Index and Offset	PC@(d,ix)	111	011	PC rel. +Ri +d3
12. Immediate Addressing	#xxx	111	100	#xxx
13. Quick Immediate	#xx	---	---	----
14. Implied Addressing	----	---	---	----

MOVE.W 8(PC),D0 {303A} Opword
{0008} Extension

Memory

Address	Data
1000	30
1001	3A
1002	00
1003	08
1004	04
1005	05
1006	06
1007	07
1008	08
1009	09
100A	0A
100B	0B
100C	0C
100D	0D
100E	0E
100F	0F

Before execution		After execution	
PC	00001000	PC	00001004
D0	00000000	D0	00000A0B

Program Counter With Index and Displacement

The program counter with index and displacement addressing mode allows the use of either an address register or data register as an index register, and an eight bit displacement. The effective address is composed of the sum of the index register, the PC, and the displacement.

The extension word for this addressing mode is given below.

15	14 13 12	11	10	9	8	7 6 5 4 3 2 1 0
D/A	Register	W/L	0	0	0	Displacement Integer

D/A — Bit indicates the index register: either a Data register or Address register.
0 — Data register
1 — Address register

Register — The register number of the index register.

W/L — This bit indicates the operand size of the index register.
0 — Sign extended lower order word integer in Index register.
1 — The long word value in the Index register.

If D0 is to be loaded with the data located at a location defined by the PC, the contents of the index register A1 and an offset of 4, then the following instruction would be used. *(Continued on page 122)*

MOVE.W 4(PC,A1.W),D0 {303B} Opword
{9004} Extension

Memory

Address	Data
1000	30
1001	3B
1002	90
1003	04
1004	04
1005	05
1006	06
1007	07
1008	08
1009	09
100A	0A
100B	0B
100C	0C
100D	0D

Before execution		After execution	
D0	00000000	D0	00000809
A1	00000002	A1	00000002
PC	00001000	PC	00001004

A Programming Example

When a fast Fourier transformer (FFT) is calculated (see references) the results are generally scrambled. To establish the proper order of the data you must bit reverse the address to the transformed data output of the FFT. All of the mathematical operations that are required to implement the FFT are available in the higher level languages. However, with all of the addressing modes of the 68000 and the ease with which address registers can be manipulated, it would seem that the bit reversal could be done easily in machine language.

The basic procedure used to form the bit-reversed word is to shift the original word one bit position to the right and check the carry bit. If the carry bit is set, one is added to D0; if the carry bit is zero, then D0 is shifted left. This process continues until D2 is decremented to zero. The results are left in the D0 register. The program that accomplishes this is given below.

```
    CLR D0          4240      Clear D0 for result
    ADDQ#$1,D2      5242      Add one to bit count
L02 SUBQ #1,D2      5342      Decrement loop counter
    BEQ L0          67000012  Get out of loop
    LSR #$1,D1      E249      Shift bits right
    BCC L01         64000008  If Carry bit 0 L01
                              If Carry bit 1 go on
    LSL #$1,D0      E348      Shift previous bit one
                              place to the left
    ADDQ #$1,D0     5240      Add 1 to D0
    BRA L02         60EE      Return to begin loop
L01 LSL #$1,D0      E348      Shift 0 into least
                              significant bit position
    BRA L02         60EA      Return to beginning of
                              program
L0  NOP             4E71
LE  BRA LE          60FE      A method of ending the
                              program. This statement
                              could have been a RTR
                              if the program were
                              called as a subroutine.
```

When this program is executed it expects to find the number of bits to be reversed in D2 and the value of the address to be converted in D1.

You may contact Professor Hootman at the University of North Dakota, Dept. of Electrical Engineering, University Station, Grand Forks, ND 58202.

Conclusions

The instruction set of the 68000 is extremely powerful and flexible. There are many programming software aids, such as the CHK instruction and the ability to use the data and address registers as index registers, that have been designed into the 68000. The 68000 will not try to execute any instruction that does not have a bit pattern consistent with a recognized opcode, and the processor will indicate such a problem by going into the exception processing state.

The hardware and software implemented in the 68000 will allow a multiple-user mode of operation as well as support high-level languages. Systems that use the 68000 will undoubtedly be used in multiple-user word processing environments and in small business environments such as grocery stores, chain stores, and department stores. Another area where the 68000 will be used is the distributive processing environment. The 68000 would be ideal as a central processor for an oil refinery where control of the refinery would be distributed throughout the refinery area.

The 68000 most likely will be upgraded from a 16-bit to a 32-bit microprocessor. Expandability of the instruction set also exists because the processor is microprogrammed, providing plenty of room for expansion. This potential of the 68000 to be expanded and upgraded means it will exist in some form for quite a while. The 68000 is a powerful machine and is well worth the effort to learn.

References

1. "16-Bit Microprocessor User's Manual," Third Edition. Prentice-Hall, Inc.; Englewood Cliffs, NJ 07632.
2. L. J. Scanion, "The 68000: Principles and Programming," Howard W. Sams and Co. Inc.; 4300 West 62nd St., Indianapolis, IN 46268.
3. G. Kane, "68000 Microprocessor Handbook," OSBORNE/McGraw-Hill; 630 Bancroft Way, Berkeley, CA 94710.
4. Brigham, E.O., "The Fast Fourier Transform," Prentice-Hall Inc.; Englewood Cliffs, NJ 07632.

The information for this series was compiled with the assistance of Motorola, Inc.

MICRO

D&N MICRO PRODUCTS, INC.

3702 N. Wells St.
Fort Wayne, Ind. 46808
(219) 484-6414

TERMS $3.00 shipping. Foreign orders add 15%. Indiana residents add 5% sales tax.

COMPUTER

MICRO-80 COMPUTER
Z-80A CPU with 4Mhz clock and CP/M 2.2 operating system. 64K low power static memory. Centronics parallel printer port. 3 serial ports. 4" cooling fan. Two 8" single or double sided floppy disk drives. IBM single density 3740 format for 243K or storage, double density format for 604K of storage. Double sided drives allow 1.2 meg on each drive. Satin finish extruded aluminum with vinyl woodgrain decorative finish. 8 slot backplane, 48 pin buss compatible with OSI boards.

MODEL 80-1200 **$2995**
2 8" Single sided drives
MODEL 80-2400 **$3495**
2 8" Double sided drives

MICRO-65 COMPUTER
6502 CPU with 2Mhz clock and DOS-65 operating system. 48K of low power static memory. 2 serial ports and 1 Centronics parallel port. 2 8" single or double sided drives. Satin finish extruded aluminum with vinyl woodgrain finish. 8 slot backplane, 48 pin buss compatible with OSI. Will run OSI 65D and 65U software. Includes Basic E/65 a compiled BASIC for 6502 CPU.

MODEL 65-1 **$2995**
2 8" Single sided drives
MODEL 65-2 **$3495**
2 8" Double sided drives

BP-580 8 Slot Backplane ..... **$ 47**
OSI 48 pin Buss compatible

MEM-CM9 MEMORY/ FLOPPY CONTROLLER
24K memory/floppy controller card uses 2114 memory chips, 1 8K and 1 16K partition. Supports OSI type disk interface

24MEM-CM9 .... **$325**
16MEM-CM9 .... **$260**
8MEM-CM9 .... **$180**
BARE MEM-CM9 .... **$ 50**
Controller on assembled unit add .... **$ 90**

BIO-1600 Bare IO card .... **$ 50**
Supports 8K of memory, 2 16 bit parallel ports, 5 serial ports, with manual and Molex connectors.

PRINTERS

Okidata
ML82A, 120 cps, 10" **$409**
ML83A, 120 cps, 15" **$895**
ML84 Parallel, 200 caps, 15" . **$1150**

C. Ioth
8510AP Prowriter, parallel ... **$419**
120 cps, correspondence quality
8510APD Prowriter, serial **$585**
F10-40PU Starwriter, parallel **$1319**
Letter quality daisy wheel
F10-40RU Starwriter, serial .. **$1319**
F10-55PU Printmaster **$1610**
parallel, Letter quality daisy wheel
F10-55RU Printmaster, serial **$1610**

DISK DRIVES AND CABLES
8" Shugart SA801 .... **$385**
single sided
8" Shugart SA851 **$585**
double sided
FLC-6 6 ft cable from D&N **$69**
or OSI disk controller to 8" drive
5¼" MPI B51 disk drive with .. **$450**
cable, power supply and cabinet. Specify computer type.
FLC-5¼ cable for connection . **$75**
to 5¼ drive and D&N or OSI controller, with data separator and disk switch. Specify computer type

HARDWARE

OSI COMPATIBLE

IO-CA10X Serial Printer Port .. **$125**
Specify Device #3 or #8
IO-CA9 Parallel Printer Port .. **$150**

CMOS-MEM
64K CMOS static memory board, uses 6116 chips, 3 16K, 1 8K and 2 4K blocks, Partitionable for multi-user, OSI type disk controller, 2 IO mapped serial ports for use with D&N-80 CPU. Ideal way to upgrade from cassette to disk.

64K CMOS-MEM **$500**
48K CMOS-MEM **$405**
24K CMOS-MEM **$260**
16K CMOS-MEM **$210**
BARE CMOS-MEM .... **$ 50**
Controller add . **$ 90**
2 IO mapped serial ports add. **$125**
on assembled memory board

Z80-IO 2 IO mapped serial **$160**
ports for use with D&N-80 CPU card
FL470 Disk Controller **$155**
5¼ or 8" drive

STANDARD CP/M FOR OSI

D&N-80 CPU CARD
The D&N-80 CPU allows the owner of an OSI static memory computer to convert to Industrial Standard IBM 3740 single density disk format and CP/M operating system. Double density disk operation is also supported for 608K of storage on an 8" diskette. When used with a 5¼" disk system 200K of storage is provided. Optional parallel printer and real time clock. Also available for polled keyboard and video systems. Compatible with C2, C3, C4 and 200 series OSI computers.

INCLUDES CP/M 2.2

D&N-80-1	Serial 8" disk	**$595**
D&N-80-2	Video 5¼" disk	**$595**
D&N-80-3	Video 8" disk	**$595**
Option 001	Parallel printer and real time clock.	**$ 60**

HARD DISK DRIVER **$140**
Allows D&N-80 CPU board to control OSI 40 or 80 meg hard disk unit. Will not destroy OSI files. Will also allow for a true 56K CP/M system. Specify 40 or 80 meg drive.

BUSS TRANSFER **$135**
Allows for D&N-80 and OSI CPU to be in the computer at the same time. Toggle switch provides for alternate CPU operation.

DISK TRANSFER **$100**
Utility program to transfer OSI CP/M format disk to IBM 3740 single density format. Will also transfer IBM to OSI format.

SYSTEM HARDWARE REQUIREMENTS
D&N-80 CPU, D&N FL470 or OSI 470 controller, 48K memory at 0000-BFFF, 4K memory at D000--DFFF, two disk drive cables.

FORMAT TRANSFER **$15**
You supply software on 8" diskette D&N will transfer OSI CP/M format to IBM 3740 CP/M format. Can also transfer IBM 3740 CP/M format to OSI CP/M format. Original diskette returned.

Circle No. 76

INPUTTING FUNCTIONS ON THE APPLE

by John D. Rippon

y = 3.14*X^2

Y = a*sin(b*x+c)

y = (1+r)^X

The machine code program in listing 1 allows a single-variable function to be input during the running of a BASIC program. As written, the routine assumes that the function concerned is referred to ahead of any other functions, strings, or variables in the BASIC program. This can be ensured by inserting a DEF FN statement at the top of the BASIC program.

The function is input as a string and then the machine-code routine is called. The function can be referred to within the BASIC program by its name in the DEF FN statement in the normal manner. Notice that it is not necessary to insert a function in the DEF FN statement, but the "=" sign is required.

Referring to the machine-code listing, the string residing in the input buffer ($200-) is tokenized by BASIC subroutine $D559 and then transferred to that section of memory labelled STORAGE - following the machine-code routine. The contents of the first function pointer following the BASIC end-of-text (vectored at $69, $6A) is then altered to point to STORAGE.

Listing 2 is a demonstration program in which a function is entered on line 50 and evaluated on line 80.

John Rippon is head of mathematics and physics at Taita College, New Zealand, where he uses a C1P and an Apple to introduce pupils to microcomputing. You can reach him at 32 Tilbury St., Lower Hutt, New Zealand.

Listing 1

```
                    *       Inputting Functions
                    *       on an Apple Computer

                    *       John Rippon

                    *       Copyright (C) 1983
                    *       by  MICRO Ink

                            START
                            ORG   $300

                    *       EQUATES

                    VARTAB  GEQU  $69
                    TXTPTR  GEQU  $B8
                    RETURN  GEQU  $8D
                    INPUT   GEQU  $200
                    TOKEN   GEQU  $D559

0300 A5B8                   LDA   TXTPTR
0302 48                     PHA
0303 A5B9                   LDA   TXTPTR+$1
0305 48                     PHA
0306 A900                   LDA   #INPUT
0308 85B8                   STA   TXTPTR
030A A902                   LDA   /INPUT
030C 85B9                   STA   TXTPTR+$1
030E 2059D5                 JSR   TOKEN
0311 68                     PLA
0312 85B9                   STA   TXTPTR+$1
0314 68                     PLA
0315 85B8                   STA   TXTPTR
0317 A200                   LDX   #$00
0319 BD0002         AGAIN   LDA   INPUT,X
031C 9D3203                 STA   STORAGE,X
031F C98D                   CMP   #RETURN
0321 F003                   BEQ   END1
0323 E8                     INX
0324 D0F3                   BNE   AGAIN
0326 A002           END1    LDY   #$02
0328 A932                   LDA   #STORAGE
032A 9169                   STA   (VARTAB),Y
032C C8                     INY
032D A903                   LDA   /STORAGE
032F 9169                   STA   (VARTAB),Y
0331 60                     RTS
0332                STORAGE DS    $80

                            END
```

Listing 2

```
10 REM Demo for Inputting Functions
20 REM by John Rippon
30 DEF FN Y(X) =
40 PRINT CHR$ (4)"BLOAD RIPPON.BIN"
50 INPUT "FN Y(X) = ";Y$
60 CALL 768
70 INPUT "X = ";X
80 PRINT "FN Y(X) = " FN Y(X)
```

MICRO

MICRO™

New Publications

Introduction to Electronic Speech Synthesis, by Neil Sclater. Howard W. Sams & Co., Inc., 4300 62nd Street, Indianapolis, IN 46268, 1983, 134 pages, paperback.
ISBN 0-672-21896-8 $8.95

CP/M Simplified, by Jeffrey R. Weber. Weber Systems Incorporated, 8437 Mayfield Rd., Cleveland, OH 44026, 1982, 316 pages, paperback.
ISBN 0-938862-04-9 $13.95

VIC-20 User Guide, by John Heilborn and Ran Talbott. Osborne/McGraw-Hill, 2600 Tenth Street, Berkeley, CA 94710, 1983, 388 pages, paperback.
ISBN 0-931988-86-1 $14.95

Microcomputer Experimentation with the Synertek SYM-1, by Lance A. Leventhal. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1983, 500 pages, paperback.
ISBN 0-13-580910-X $19.95

Stimulating Simulations for the VIC, by C. W. Engel. Hayden Book Co. Inc., Rochelle Park, NJ, 1983, 91 pages, paperback.
ISBN 0-8104-5173-5 $6.50

The Apple II Circuit Description, by Winston D. Gayler. Howard W. Sams & Co., Inc., 4300 West 62nd St., Indianapolis, IN 46268, 1983, 172 pages, plus schematics, paperback.
ISBN 0-672-21959-X $22.95

TRS-80 Color Computer Interfacing, with Experiments, by Andrew C. Staugaard, Jr. Howard W. Sams & Co., Inc., 4300 West 62nd St., Indianapolis, IN 46268, 1983, 203 pages, paperback.
ISBN 0-672-21893 $14.95

Peripherals, Etc., National magazine whose editorial thrust is at hardware! 1797 Capri Ave., Mentone, CA 92359, Premier Issue July 1983.
Sub. Rate 1 yr./$22.50
2 yrs./$39.50

COMAL Handbook, by Len Lindsay. Reston Publishing Company, Inc., A Prentic-Hall Company, Reston, VA 22070, 1983, 310 pages, paperback.
ISBN 0-8359-0878-X $18.95

Atari Programming with 55 Programs, by Linda M. Schreiber. Tab Books, Inc., Blue Ridge Summit, PA 17214, 1982, 244 pages, paperback.
ISBN 0-8306-1485-0 $13.95

Business Management Tools for the Apple II and IIe

Microsoft Corporation announces the introduction of software management tools that design and quickly build customized **Multiplan** electronic worksheets. The **Multi-Tool** expert systems create finance and accounting worksheets in Multiplan for business managers and professionals. Extensive tutorial documentation, written by leading finance and accounting authorities, provides expert guidance for the user. Two software packages are now available: The **Multi-Tool Budget** expert system for budget planning and control and the **Multi-Tool Financial Statement** expert system for performing financial statement ratio analysis. Each package consists of the program disks, a user manual, and a case study folder.

The program disks contain a demonstration that teaches new users how to use the Multi-Tool expert system, programs that prompt the user for information for building the Multiplan worksheets, and example cases — interactive case studies that help the user understand how a hypothetical company would develop the worksheets.

Price is **$150.00** for the Budget expert system and **$100.00** for the Financial Statement expert system. For further information write to Microsoft Corporation, 10700 Northup Way, Bellevue, WA 98004; (206) 828-8080.

Format II Enhanced Version

Format II Enhanced Version word processing software for Apple II+, Apple IIe, Franklin, and workalike computers is a powerful word-processing system that permits extremely flexible form-letter preparation, label, printing, and more. The data base manager in Format II Enhanced Version uses the familiar label-for-variable swapping technique. A new version of the base document is printed for each selected record in the "mailing list" data base, customized with the appropriate information from each record being substituted each time. In addition to preparing form letters from mailing lists, this also permits the easy generation of labels and substantially repetitive reports. The user can choose to generate a document for every record in the data base or only for records that meet selected criteria. The text files created by Format II Enhanced Version are now standard DOS 3.3 files, providing compatibility with off-the-shelf spelling, proofreading, and communications programs.

Format II Enhanced Version requires an 80-column Apple II+, Apple IIe or Apple workalike with DOS in ROM, at least 64K, and one disk drive. The availability of a second drive will enhance its operation but is not absolutely required. Price is **$150.00** at participating Kensington Microware dealers. For additional information contact Kensington Microware Ltd., 919 Third Avenue, New York, NY 10022; (212) 486-7707.

DB Master Version Four

DB Master Version Four allows the user to have up to 100 characters in an alphanumeric field. For greater ease of use, a Field Editor has been included. This feature is similar to a word processor, allowing the user to add, insert, and delete text anywhere in a field without having to re-enter the entire field. When editing records the user can modify a record and then store it either as a new record or as the revised original record. Other DB Master Version Four features include easier file and report format design, improved screen prompting, and reuseable sort diskettes. A short tutorial on file creation has been added to the program to assist the first-time user. To accommodate these and more features, a second program diskette has also been added.

DB Master Version Four is compatible with files made with previous versions and with the DB Master Utility Paks and the new Stat Pak. Hardware requirements include an Apple II Plus with 16K RAM card or an Apple IIe and one to four disk drives (two are recommended). A printer is also recommended to make use of the extensive report generator capabilities.

Suggested retail price for DB Master Version Four is **$350.** Registered owners of DB Master Version Three will be offered a trade-in towards Version Four at a reduced price. Contact Stoneware, Inc., 50 Belvedere Street, San Rafael, CA 94901; (415) 454-6500.

Software Catalog *(continued)*

Apple II Editor/Assembler

CUMSEA is a professionally designed Apple II editor/assembler ideally suited for use in teaching 6502 assembly language and for use in advanced computer literacy courses. The system includes a 9" x 6" vinyl binder containing a diskette with eight programs, three test files, two demonstration files, a typeset 83-page user manual, and a diskette replacement-warranty card. All programs are self-protecting, user friendly, flexible, and easy to use. Two of the eight programs are not available for student use; they are used to 'foolproof' the remaining six programs and to provide a MENU from which these six programs may be selected. The menu-selectable programs consist of an editor, an assembler, a utility program, and three format-converting programs. In addition to in-depth operating information, the manual contains a 5-page table of contents, a self-teaching familiarization section, five appendicies, and a 10-page cross-referenced index.

Available for **$26.50** (US) or **$32.00** (Canada). CUMSEA may be ordered at the special price for teacher and educational use only, FOB Edmonton. This is a promotional offer available only from Custom Micro Systems Ltd., 16921 108 Street, Edmonton, Alberta, Canada, T5X 3B2.

Graphics Writer for OSI, VIC-20

Machine-Language Graphics Writer for OSI C1P and C4P and the Commodore VIC-20 writes a machine-language program to draw the same picture that you make using the draw routine. You can draw pictures with any of the characters except for the 32-decimal blank space. Once a picture is drawn, the computer will generate a machine-language program and store it in high memory. The routine can be saved on cassette tape and loaded back in later. You can save the routine in DATA statements if you wish. On the VIC-20 and OSI C4P, color is also used. Using the machine-language routine, you can display and erase the picture you drew on the screen at very high speeds.

Price is **$5.00** and includes cassette and instructions. Contact Brian Zupke, B.C. Software, 5152 Marcella Ave., Cypress, CA 90630.

THE SCREEN PRINTER

THE SCREEN PRINTER, developed by SMART Systems, Incorporated, is an Apple Pascal procedure that will transfer the exact contents of the screen onto any printer. Either the primary or secondary text screen may be printed, or both side-by-side for 80-column display. The Videx Videoterm 80-column card is also supported. An Apple II+ with Apple Pascal and any printer are required.

The price is **$25.00** and includes program diskette, documentation, postage and handling. For further information please contact Janice Bertozzi at SMART Systems, Inc., 499 Sulky Lane, Frederick, MD 21701, (301) 694-8307.

Passive Voice Module

The **Passive Voice Module** of the *English Structure Series* provides practice in using the passive voice. The module consists of a screening exercise and three practice exercises, each with a different format. Each exercise features a random selection of problems, allowing for multiple, non-repetitive practices. Also included in the program is a system for keeping records of each student's performance. Accompanying the module is a teacher's manual, which includes a complete listing of all verbs, sentences, and paragraphs used in the program, as well as descriptions of visual displays. The manual gives precise instructions on how to use the program and the record-keeping system. This program is compatible with the Apple II computer, 48K with disk.

Price is **$42.50.** Contact River Bend Software, P.O. Box 637, Atchison, KS 66002-0637.

Experiments in Human Physiology

Experiments in Human Physiology is an innovative new microcomputer package from HRM Software, which allows science and psychology students to measure and record fine physiological parameters. Developed by Dr. Robert F. Tinker at Technical Education Research Centers, the kit is designed for use on the Apple II, II+, or IIe with 48K RAM, DOS 3.3, and Applesoft BASIC. Among the topics students can investigate are homeostasis, biofeedback, the physiology of exercise, stress, and sleep. The kit can also be used for experiments in psychology. The package includes everything needed for ten experiments: a disk containing the experiments and instructions, three probes (no electrical contact to subjects is required), and an interface box that connects to the Apple game-paddle ports without interfering with normal game-paddle operation.

The package costs **$250.00,** and is available on a firm order basis only with a 30-day full refund option. Orders may be placed by writing on school letterhead to HRM Software, 175 Tompkins Avenue, Pleasantville, NY 10570 or by calling toll-free (800) 531-2050. (In New York or Canada, call collect (914) 769-7496.) Order# 502-00-SABH)

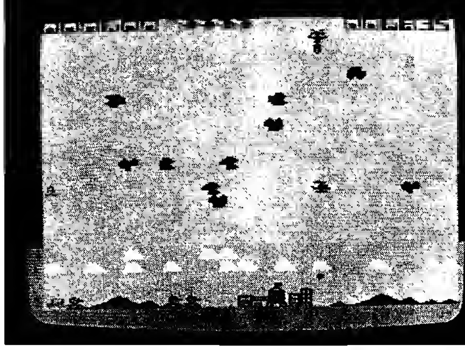
(Continued on page 130)

New Games for the VIC-20

Tronix Publishing, Inc., has introduced three fast, high-quality action video games for the Commodore VIC-20 home computer. **Deadly Skies, Gold Fever!,** and **Scorpion,** are available through Tronix's distributors and directly from retail computer stores.

Price is **$39.95** each. For more information on Tronix's new games for the Commodore VIC-20 contact Tronix Publishing Corp., 8295 S. La Cienega Blvd., Inglewood, CA 90301; (213) 671-8440.

Left: Scorpion, — a new fast-action game for the Commodore VIC-20.



Left: Gold Fever! — a new fast-action game from Tronix Publishing, Inc.

Above: Deadly Skies — a "shoot- 'em-up" action game.

Educational Software

Word Attack and **Math Blaster** are two new educational software products for the Apple and IBM. Word Attack is a vocabulary-building system that makes learning new words fun and easy. It has four educational exercises (including a fast-action arcade game) that access extensive data files of 675 words. The words and sentences illustrating usage are presented on nine different levels for students age eight through adult. Word Attack also has an editor to enter additional word lists. Math Blaster contains over 600 problems in addition, subtraction, multiplication, division, fractions, and decimals for students age 6–12. The problems are grouped in "families of facts" and can be used with four different learning activities (including a fast-action arcade game). Math Blaster also contains an easy-to-use editor that allows the student, teacher, or parent to enter new problems for use with all four learning activities.

Word Attack and Math Blaster each contain two disks and a 60-page user's manual for **$49.95.** For more information contact Davidson & Associates, 6069 Groveoak Place #12, Rancho Palos Verdes, CA 90274; Call (213) 378-7826.

[redacted]

Software Catalog *(continued)*

Micro Math

Micro Math is a new series of educational math programs for 12-16-year olds. Part of the more extensive Master Math package, Micro Math is intended for use on the Sinclair ZX81, Timex 1000, Commodore PET, and VIC-20 in schools and colleges as a computer-assisted learning aid and at home as a self-tuition and revision course. Micro Math is comprised of six program suites, each containing four programs available on two cassettes.

Price is **$50.00.** For more information contact PM International, P.O. Box 87, Buckfield, ME 04220; (207) 336-2500.

New Games from Penguin

Thunderbombs and **Crime Wave** require 48K Apple with disk drive, keyboard or joystick control. Mockingboard Speech and Sound Board option. Price: **$19.95** each. **Spy's Demise** requires 32K Atari 400 or 800 for disk or 24K Atari 400 or 800 for cassette, keyboard, joystick, or paddle control. Price: **$19.95.**

Color slides or black-and-white screen dumps are available for illustration. Please contact Mary Locke at Penguin Software, 830 4th Ave., Geneva, IL 60134; (312) 232-1984.

Mnemonic Assembler

The Cheap Assembler is a mnemonic assembler for the Apple II computer. Now, for less than the cost of a game, you can have a complete assembler/editor system at your disposal. The system features include unlimited length labels, free-field programming, two-pass RAM/disk-based assembly, text editor with ten commands including character-insert and delete, line-insert and delete, type-over modifications while viewing the preceding lines on the screen, interactive operation, and explanatory debugging messages. Edit, assemble, and run a program without ever leaving the system. Also included are a tutorial-style manual and demonstration routines on disk. There is no copy protection.

Apple II+, 48K DOS 3.3 required. Price is **$20.00** plus $4.00 postage and handling. Available from Thunder Software, P.O. Box 31501, Houston, TX 77231; (713) 728-5501.

Want to list your product in Software Catalog? Write to MICRO, P.O. Box 6502, Amherst, NH 03031.

Commodore 64 and VIC-20 Products

Computer Marketing has announced distribution of several new products for the Commodore 64 and VIC-20 computers. Designed to be used with a monitor, **Video Pak 80** lets the Commodore 64 users switch from 40- to 80-column screen format. **Z-80 Video Pak** provides the convenience of an 80-column screen format and the power of CP/M capability. Available for both the Commodore 64 and the VIC-20, the Printer Interface is supplied with a cable to connect to RS-232-interfaced printers and other devices, such as modems.

For the VIC-20, the **Video Pak Cartridge** increases the video display from 23 lines of 22 characters to the industry standard 24 lines of either 40 or 80 characters. The 16K Memory Cartridge is a low cost, high quality way of increasing the memory of the VIC-20, and the Expansion Chassis gives the VIC owner the capability of using any four compatible cartridges simultaneously.

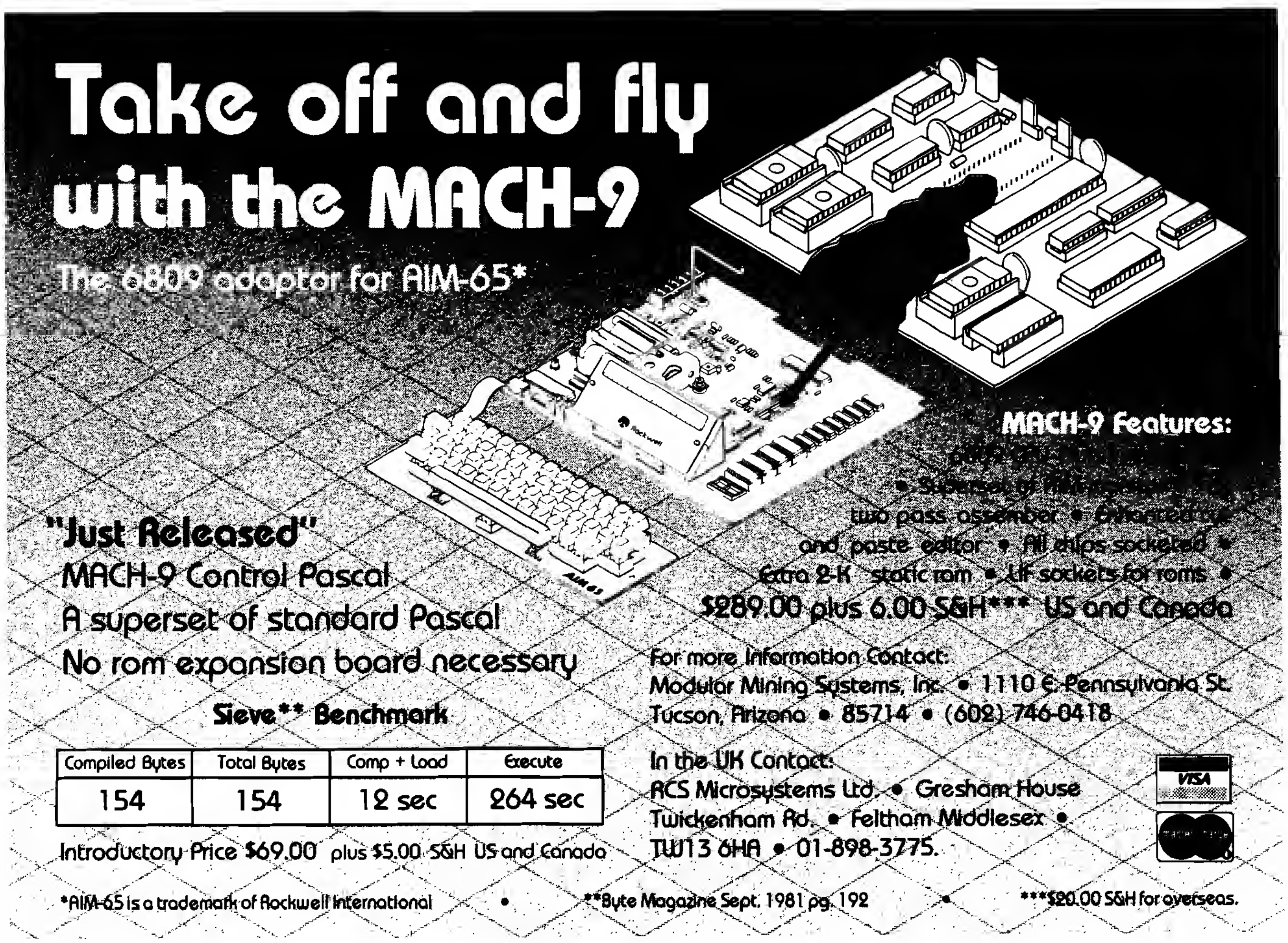
For more information contact Computer Marketing Services, Inc., 300 W. Marlton Pike, Cherry Hill, NJ 08002; (609) 795-9480.

Instant Recall

Instant Recall is an easy-to-use freeform filing system for the Apple II and Apple II+ disk system with at least 48K. It has been designed to avoid the necessity of user preparation of fields and formats. Instead, information can be entered on a fresh screen in much the same way as on a blank sheet of paper or index card. Any variety of information can be intermixed on a single disk file, or separate disk files can be established for specific purposes. Information from even a long file can be retrieved in, at most, two seconds; and the initial loading time for a file is five seconds or less.

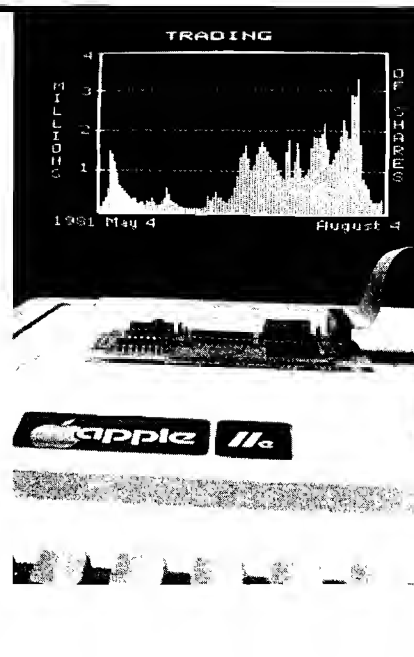
Price is **$59.95** and includes an Instant Recall tutorial, a command quick-reference card, specimen screens, separate keyboard diagrams highlighting special key functions in each mode, and a complete index. Available from Sams Software dealers. For more information contact Howard W. Sams & Co., Inc., 4300 West 62nd St., P.O. Box 7092, Indianapolis, IN 46206; (317) 298-5708.

MICRO™



Circle No. 89

A Parallel Printer Interface for Apple

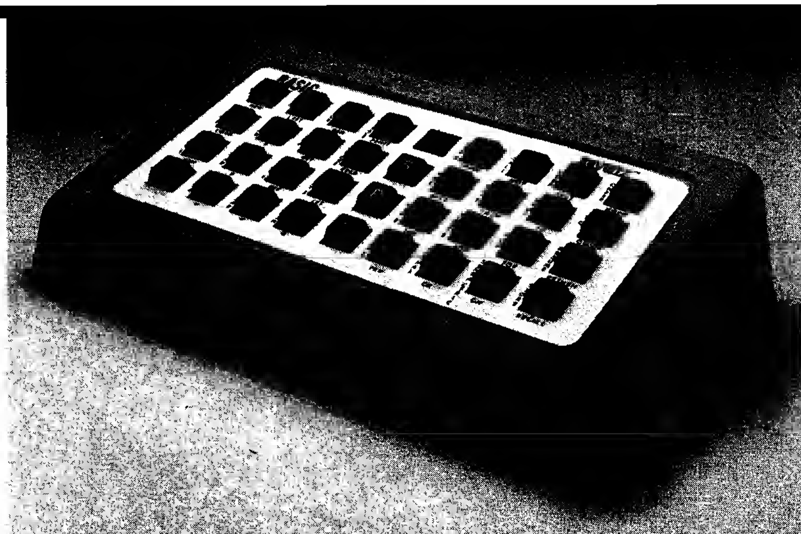


SSM Microcomputer Products, Inc. announces the APPIC/G, a new parallel printer interface for the Apple II and Apple IIe. The package includes an APPIC interface card, cable, and Graph-It graphics software. The APPIC parallel interface works with any standard parallel printer. Graph-It, a complete graphics dump program, allows you to do a screen dump to any of the following printers: Apple Dot Matrix, MPI, IDS, NEC, Okidata, Silentype, Epson, Trendcom, Anadex, C. Itoh, and Centronics.

While other parallel cards require purchase of additional firmware to support different printers, with the APPIC/G you do not change anything — simply choose your printer from a menu and print.

With Graph-It you can print graphs, charts, equations, grids, logos, titles, and hi-res Apple graphics. You can also magnify, print normal or reverse image, do page centering and cropping, or insert graphics, superscripts, and subscripts into any Applewriter II file.

The SSM APPIC/G package is **$129.00.** Available from SSM Microcomputer Products, Inc., 2190 Paragon Drive, San Jose, CA 95131; (408) 946-7400.



The KeyWiz VIP — A New Auxiliary Keyboard

Creative Computer Peripherals, Inc., has announced its new auxiliary keyboard called the **KeyWiz VIP** (Very Intelligent Peripheral). It consists of 31 user-programmable function keys housed within a new injection-molded color-coordinated case. Each of the 31 keys may be easily programmed with up to eight characters each and programmed again while using the shift key giving the user 62 user-defined keys. Four such keyboards are stored in the devices memory, easily accessed anytime, making 248 preprogrammed keys available to the user at the touch of a button. A 7-segment LED displays which one of the four keyboards you're using. Each key may be redesignated over and over again, even in the middle of a program, as your needs change. KeyWiz VIP is independent of any software and can be ordered for the Apple II+, Apple IIe, Franklin Ace, or TRS-80 Model III.

For more information write Creative Computers, Aztec Environmental Center, 1044 Lacey Road, Forked River, NJ 08731; (609) 693-0002.

(Continued on page 136)

New Apple Detachable Keyboard by AMKEY

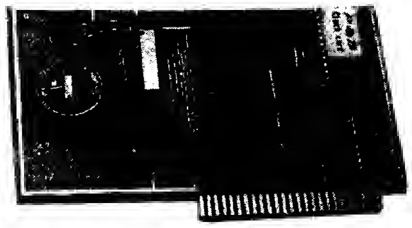
The **PRO-100** is an intelligent, detachable, capacitance keyboard with enclosure for use with the Apple II or Apple II+. It offers 100 keys supporting all existing Apple functions plus horizontal and vertical cursor movement, separate number pad with enter key, auto-repeat, relocated reset key, CAP's lock key, power-ON indicator, upper/lower case (rev. 7+), 22 VisiCalc keys, 25 Apple BASIC keys, and 18 programmable keys.

The PRO-100 keyboard package is compatible with existing Apple peripherals. It includes AMKEY's high quality capacitance 100 key keyboard with enclosure and 6-foot interface cord, upper/lower case chip, boot diskette, and user-installation instructions.

Price is **$265.00**, including shipment and handling. For more information contact Mr. Robert Dimodana, Vice President, AMKEY, INC., 220 Ballardvale Street, Wilmington, MA 01887; (617) 658-7800.

Hardware Catalog *(continued)*

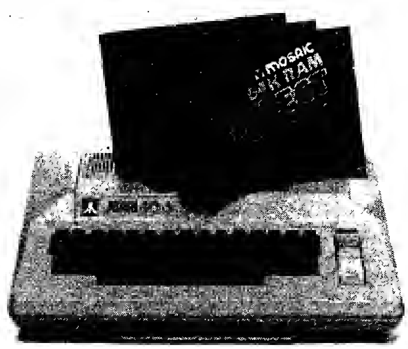
Low-Cost Clock Card Available for Apple



The new **dat•a•clock©** multi-function plug-in card designed by P & B Research Consultants is fully compatible with Apple computers including the Apple II, Apple II+, and Apple IIe. Purchaser has the option of ordering the dat•a•clock either assembled by the manufacturer or in a do-it-yourself kit.

Easy to use, the msm 5832 has data, month, and year capability, and the on-board battery has a two to three-year life. There is also an externally accessible EPROM.

Price is **$85.00** (assembled) or **$55.00 (kit** with easy-to-follow assembly instructions) plus **$2.00** shipping and handling. Send check or money order to P & B Research Consultants, 231 East Grand Blvd., Detroit, MI 48207. For additional information contact Art Potter, (313) 259-5951.

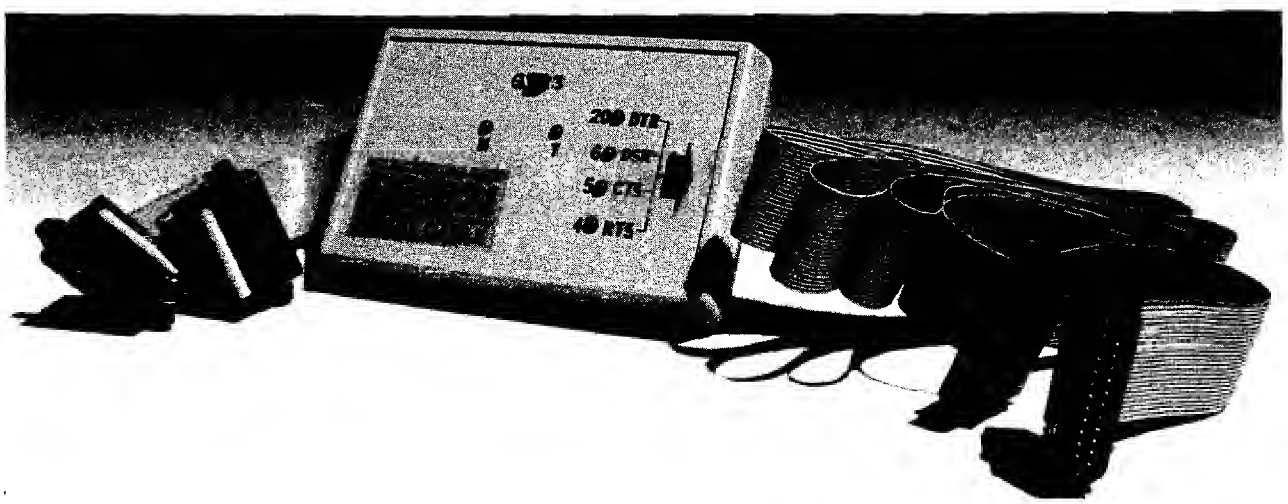


New Life for the Atari 800

The **Mosaic 64K RAM Select** is a new board that will plug into both the Atari 400 and 800. Atari 800 owners can plug up to three Select boards into their computer for 196,608 bytes of useable RAM. The Mosaic 64K RAM Select is totally bus-compatible for use with Atari 16K and/or Mosaic 32K RAM boards. Atari 800 owners can use the board to simulate the Atari 1200 architecture or configure the board for super powerful bank selection. The 64K Select is compatible with both 8K and 16K ROM cartridges and installs without solder. Used with the Mosaic Adapter, 48K Atari 800 owners will have 112K RAM.

For more information write Mosaic Electronics, Inc., P.O. Box 708, Oregon City, OR 97045.

New RS-232 Interface Cable



IQ Technologies, Inc. announces its new "intelligent" RS-232 interface cable, the **SC821 SMART CABLE,** which instantly hooks up any computer to any peripheral with the flick of a single switch. The unique onboard logic circuitry looks at the RS-232 interface on both the computer and the peripheral and then correctly connects the interfaces. SMART CABLE eliminates the need for cable design, "breakout boxes", and the need to maintain a large inventory of custom cables to ensure correct connection. It is invaluable for equipment demonstrations, systems integration, equipment leasing, engineering, field service, and debugging.

The SMART CABLE connects all handshake lines used in a specific application in addition to CTS, DTS, DTR, and DSR. It functions at baud rates up to 19,200. Indicator lights point out which device is disabling data transfer in the event that a hardware or software problem exists. SMART CABLE is completely transparent to baud rate, word length, error, and data codes.

Price is **$245.00** complete. Order model SC821 from IQ Technologies, Inc., 11811 N.E. First Street, Suite 308, Bellevue, WA 98005; (206) 451-0232. TWX 910-443-2308. MICRO™

MICRO™

Reviews in Brief

Product Name: **Platinum Worksaver**
Equip. req'd: TRS-80 Color Computer
Price: $30.00 plus shipping
Manufacturer: Platinum Software
P.O. Box 833
Plattsburgh, NY 12901

Description: *Platinum Worksaver* is a single-key data-entry system and screen editor for the Color Computer. The software provides automatic line numbering and an insert mode. Keys can be redefined with new definitions stored with the program. A keyboard overlay is included, displaying the control key functions. A numeric mode turns the J, K, L, U, I, O, 7, 8, 9, and 0 keys into a numeric keypad.

Pluses: The program provides several useful new capabilities for CoCo. The software is disk compatible, and provides an automatic "PCLEAR 0" upon execution, which gives an extra 1.5 of memory for program storage if no graphics are required; the screen editor functions in program mode, allowing array editing from within a program; dynamic editing allows programs to be modified without destroying data already created; BASIC lines can be split or joined; and the computer can be shifted into high or low speed with a two-key command.

Minuses: A minor fault is that the user must remember not to execute a NEW command; a DELO- is used instead.

Documentation: A 28-page manual is provided that steps the user through the powerful new keyboard. In addition, an array editor program is included that demonstrates the ease and power of dynamic input.

Skill level required: A knowledge of BASIC programming is required, since it is a programmer's utility.

Reviewer: John Steiner

Product Name: **Color Diskette Repair**
Equip Req'd: TRS-80C Color Computer with disk and 16K memory
Price: $31.95
Manufacturer: Computerware
Dept. C, Box 668
Encinitas, CA 92024

Description: This product allows the user to examine files as written on disk by the Color Computer, change those files, and rewrite them to the same sector or a new sector. However, if a file has been erased or damaged, normal DOS commands cannot read them. *Color Diskette Repair* helps restore such files. The disk also can read a binary file and extract the start, end, and transfer addresses of a file to facilitate transfer to tape (FIND). VIEW allows you to examine and print out ASCII files from disk: MASTER LISTER prints a disk's directory, and MASKILL allows you to delete unwanted files from a disk quickly.

Pluses: Low cost for a disk program. Several useful features for the advanced disk user.

Minuses: MASTER LISTER should list track and sector linkage information, or else REPAIR should display the file headers (bytes 0-55) that contain the linkage information. Also, VIEW should handle all file types instead of just ASCII files. (In addition, the copy I reviewed had a bug in the granule-counting routine, which requires that the user add two to the granule number shown in the directory if the file is located above granule $22 (34 decimal.))

Skill level required: This product is aimed toward the serious disk user and requires a good understanding of disk operating principles.

Documentation: Nine pages of instructions. However, directory access is not clearly spelled out; when first loaded, REPAIR defaults to the directory, but this is not stated. Without this clue I found it difficult to get started.

Reviewer: Ralph Tenny

Product Name: **Multiploy**
Equip. Req'd: Apple II+
Price: $19.95
Manufacturer: Reston Publishing Co.
11480 Sunset Hills Road
Reston, VA 22090
Author: Paul Coletta

Description: This is an arithmetic drill (whole number add, subtract, multiply, divide) disguised as an arcade game. Shoot down the attacking problem ship (by answering the problem correctly) before it shoots you. You can make the game more exciting by choosing to have the problems attack at lightening speed.

Pluses: There are three levels of difficulty. Any missed problems are highlighted at the end of the game. The program saves the highest score and has an option for less sound. Good for classroom application.

Minuses: Does not record individual achievement.

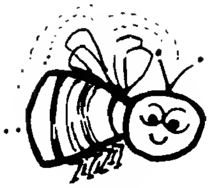
Documentation: A booklet includes in-depth instructions.

Skill level required: A game parents can do with their children ages 6-14. An adult would have to explain the instructions to a young child.

Reviewer: Mary Gasiorowski

(Continued on page 140)

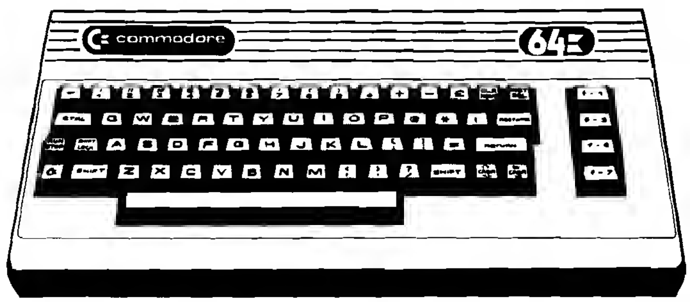
BUSIWRITER

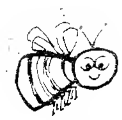


BUSIWRITER A Honey of a Word Processor

Why word processors?

Word processors allow the user to quickly and easily create letters, memos, notes, reports, term papers, manuals, poetry and any other written information using the memory of the computer as a pencil and paper. The computer display or terminal acts as a window through which the user views the information as it is entered. The outstanding advantage of using BUSIWRITER is that it acts not only as a pencil and paper but as a perfect eraser and automatic typewriter.

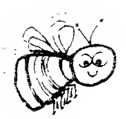




For Commodore CBM-64
Commodore 1515, 1525, Epson, C. Itoh, Qume, Diablo, NEC Spinwriter, Starwriter, Prowriter, Okidata, Microline, Gemini-10
And many more printers

BUSIWRITER The Queen Bee of Word Processors

BUSIWRITER allows the user to quickly and easily make any number of alterations to the text. BUSIWRITER will instantly reformat your text and show you exactly and continuously how the final output will appear. BUSIWRITER has more functions than any other known microcomputer word processor. With BUSIWRITER assisting in the entry of text, providing a **20 page memory** and performing an enormous number of editing/composing functions, the preparation of written data is far faster and outstandingly more accurate than if it were prepared by hand.



BUSIWRITER With the Sting Removed from the Prices

BUSIWRITER 64...............................only **$99.00** for the CBM 64

BUSIWRITER AVAILABLE NOW FROM YOUR LOCAL DEALER
(800) 227-9998
FOR THE NAME OF YOUR NEAREST DEALER

California, Canada, Alaska and Hawaii please call (415) 965-1735

Skyles Electric Works
231G South Whisman Road
Mountain View, CA 94041

Europe please contact Supersoft, Winchester House, Harrow Wealdstone, England HA3 7SJ, Tel. 01 861 1166

Circle No. 92

Reviews In Brief *(continued)*

Product Name:	**RAM/EPROM Memory Board, p/n 81-330**
Equip. req'd:	AIM, KIM, SYM or SLIM microcomputer
Price:	$99.95 Assembled w/o memory (81-330A) $49.95 Bare board (81-330B)
Manufacturer:	John Bell Engineering, Inc. 1014 Center St. San Carlos, CA 94070 (415) 592-8411

Description: This memory board, in the popular 4.5" x 6.5" card size, uses the industry standard dual 22-pin connector. The connector pinout is compatible with the expansion connector on AIM, SYM, and KIM microcomputers and is designed as a memory expansion for the SLIM microcomputer. Sixteen memory positions are designed to accept either 2716-type EPROMs or 6116-type read/write memory devices for a total of 32K bytes of memory expansion. The board uses the entire 16-bit 6502 address bus and is decoded for the address range $0000-$7FFF, with an exclusion switch to allow the $000-$7FF block to be disabled. If the existing 2114-type memory is removed from AIM, SYM, and SLIM boards, it is possible to populate this board with 6116 memory and get a 32K computer with contiguous memory from $0000-$7FFF; the combination will draw no more power than the single-board computer and, perhaps less!

Pluses: Excellent construction allows even inexperienced assemblers to assemble a bare board version.

Minuses: The documentation is minimal and an inexperienced user could have a difficult time getting the board on-line without advice.

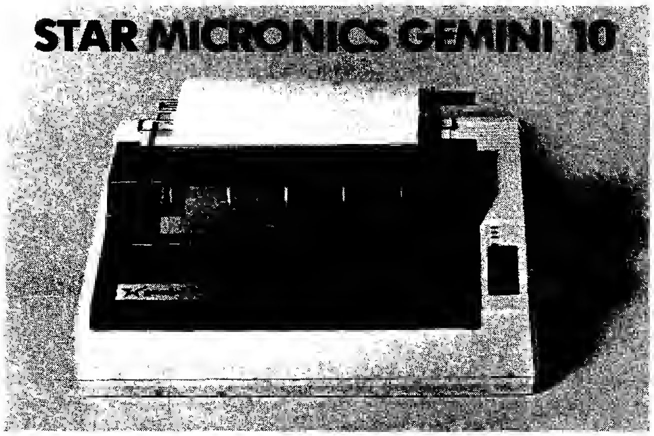
Documentation: Less than the bare minimum.

Skill level required: Even the assembled version should not be purchased by anyone inexperienced with computer hardware, unless he has a source of advice.

Reviewer: Ralph Tenny

OMEGA Ω INTERNATIONAL

SAVE ON ... COMPUTERS · MONITORS PRINTERS · PERIPHERALS · SUPPLIES*



OMEGA SALE PRICED PRINTER

• DOT MATRIX **$329** SALE ENDS 6/30/83

• Up To 160 cps • 11 x 9 Matrix • Pinfeed Platen
• Proportional Spacing • Graphics • Elite Pitch
• Centronics Parallel Interface • Internal 2 K Ram

NOW AVAILABLE FOR IMMEDIATE DELIVERY!

For Our Price...CALL 1-800-343-0873 TODAY!

SAVE ON LETTER QUALITY PRINTERS

DIABLO 620 New Low Price	929.00
DIABLO 630 w/API & cable	1,749.00
NEC 3510 SPINWRITER	1,399.00
NEC 7710 SPINWRITER	2,045.00
NEC 7730 SPINWRITER	2,095.00

DOT MATRIX PRINTER BARGAINS

C-ITOH PROWRITER 8510 AP	399.00
IDS MICROPRISM 480	549.00
OKIDATA MICROLINE 92 (NEW)	549.00
OKIDATA MICROLINE 93 (NEW)	859.00

BIG SAVINGS ON ACCESSORIES

HAYES SMARTMODEM 300 Baud	230.00
HAYES MICROMODEM II (APPLE II)	289.00
MICROSOFT SOFTCARD PREMIUM SYSTEM	459.00
ORANGE MICRO GRAPPLER +	120.00
PKASO PRINTER CARDS	129.00
RANA ELITE I (APPLE II)	299.00
SIGNALMAN MODEMS (MK I) As Low As	85.00

MONITOR SPECIALS FROM OMEGA

AMDEK 300 G	139.00
AMDEK 300 A	165.00
NEC JB1260	119.00
NEC JB1201 M	169.00
USI Pi-2 12" GREEN MONITOR	159.00
USI Pi-3 12" AMBER MONITOR	179.00

ACCESSORIES & SUPPLIES

OMEGA Has A Complete Line of Accessories & Supplies for the Apple II and many other Popular Computers by manufacturers like:
• D. C. Hayes • Microsoft • Tymac
• M & R Enterprises • Mountain Computers
• Kensington Microware • Practical Peripherals
• T.G. Products • Videx

SOFTWARE

Omega Caries Software by the following companies:
• American Business Systems • Ashton Tate
• Dakin 5 • Innovative Software • Microsoft
• Sorcim • Stoneware • Visicorp

MAGNETIC MEDIA

OMEGA Stocks Diskettes by:
• Dysan • Elephant • Maxell • Verbatim

• All Equipment Factory Fresh w/ MFT Warranty
• Prices Do Not Include Shipping Charges
• Mass. Residents Add 5% Sales Tax
• All Returns Subject To Restocking Fee

CUSTOMER PICKUP NOW AVAILABLE
334 R Cambridge St., Burlington, Mass.
(617) 229-6464

CALL TOLL FREE!
1-800-343-0873
Call Toll Free for Ordering.
All Others call (617) 229-6464

CHARGE IT!
MasterCard / Visa
WELCOME AT NO EXTRA CHARGE

* PRICES, SPECIFICATIONS AND AVAILABILITY OF ADVERTISED MERCHANDISE SUBJECT TO CHANGE WITHOUT NOTICE

UNADVERTISED SPECIALS ON • COMREX • EPSON • NEC • IDS PRISM • OKIDATA

OMEGA INTERNATIONAL

334 R CAMBRIDGE STREET, BURLINGTON, MA. 01803

MICRO™

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6809 Bibliography

140. The Rainbow 1, No. 6 (December, 1982) *(continued)*

Lewandowski, Dennis S., "Target Practice: Let's Blast Those 'Vaders,' " pg. 106-108.
A machine-language listing for a TRS-80 Color Computer game.

Selig, Paul, Jr., "Three Disk Utilities Give You Filing Help," pg. 110-111.
Listings for the Color Computer to search, print, or convert an U/L case file to all upper case.

Weiss, A. Arnold, "Unidatfl Revisited: Variations on Mir's Theme," pg. 114-123.
Modifications to a filing program for the TRS-80 Color Computer.

Witham, B.B., Jr., "Ham and CoCo; Holiday Fare for Duel Hobbyists," pg. 124-130.
Station Log program for Color Computer users having Ham Radio as a second hobby.

O'Donnell, Timothy J., "Word Search Generates Learning as Well as Fun," pg. 130-134.
A word-search puzzle generator for the TRS-80 Color Computer user.

Chamberlain, Ross, "This Graphic Printer Will Give You a New Typeface," pg. 136-141.
A program for the CoCo and the Line Printer VII.

141. System-68 Magazine, No. 5 (December, 1982)

Pentecost, Joe, "Beginning Assembly-Language Programming," pg. 9-18.
A tutorial using an introduction to machine language for the 6809-based systems.

McGee, Wayne, "Operating Systems for the EXORciser II," pg. 21-25.
A development system based on the MC6809, and a comparison of several different operating systems (FLEX, MDOS, OS9).

142. Color Computer News, Issue No. 15 (December, 1982)

Sias, Bill, "REMarks," pg. 3.
A note on the upgrading of the TRS-80 Color Computer to 64K on the revision F board.

Sias, Bill, "Mail Call," pg. 10.
A short routine for disabling the break key on the 6809-based Color Computer.

William, Old Father, "Making (CLEAR) a Bit Clearer," pg. 14-19.
Some solid information on the command (CLEAR 1000) of the 6809-based TRS-80 Color Computer.

Ryan, Joseph A., "Vertplot," pg. 20-26.
A Color Computer program for use in studying the patterns of shortwave radio antennas.

Degler, Roger L., "FLEX Corner," pg. 27-32.
Disk Operating Systems for the Color Computer.

Sterling, Jeffrey D., "CC as a Photography and Darkroom Aid," pg. 34-37.
A Color Computer program on hyperfocal distance of lenses, including a program for exposure values using polycontrast filters and a program for enlarger-exposure conversions.

Wilson, Mark E., "Draw A Circuit," pg. 38-42.
A program for the Color Computer using a prepared character set.

Hogg, Frank, "64K Korner," pg. 43-47.
Using the upper 32K to display memory, CoCo Bulletin Boards, using the full 64K RAM, FLEX vs. OS(, etc.

Button, John, "Addition of a Software-Based TIME/DAY Clock to the Color Computer," pg. 48-52
A software clock for CoCo.

Anon., "Homework Helper — Factors," pg. 53-57.
FACTORS is a program for the TRS-80 Color Computer designed to help a student with problems involving factoring.

Ventling, James, "Text Screen Graphics Using POKE and PEEK," pg. 59-61.
An article to help you make the most of your TRS-80 text screen, PEEK and POKE, etc.

Kelly, Joseph, "TIC-TAC-TOE," pg. 62-65.
A game for the 6809-based Color Computer.

Crager, John, "Copter," pg. 68-69.
A game for the Color Computer.

Santee, Charles H., "Sound Effects Generator," pg. 73-74.
A program to explore sound effects that can be produced by the PLAY statement on the TRS-80 Color Computer.

Smith, Randall, "Elementary Math," pg. 76-80.
An educational program for the 6809-based Color Computer.

Morrow, Ken, "Program Relocation II," pg. 85-88.
A utility program for the Color Computer. MICRO™

—Next Month in MICRO—

Feature:

Word Processing

- **What is word processing?**
  What features do you need in a word processor?
- **Word Processing with Pascal**
  This program provides full word-processing capabilities for your Apple Pascal Language System Editor.
- **Dvorak Keyboard Demonstration**
  The Dvorak Simplified Keyboard is faster than the standard typewriter/computer keyboard. Use this program to experiment.
- **The Selectric Word Processor**
  This conversion keyboard uses an IBM Selectric terminal as a low-cost printer.

Plus:
Our hardware series continues. This month we look at peripherals available for the Color Computer, the TI 99/4A, and Commodore machines.

Advertiser's Index

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

National Advertising Representatives

Middle Atlantic and Southeastern States:

Dick Busch Inc.
Richard V. Busch
6 Douglass Dr., R.D. #4,
Princeton, NJ 08540 (201) 329-2424

Dick Busch, Inc.
Eleanor M. Angone
74 Brookline,
E. Atlantic Beach, NY 11561 (516) 432-1955

serving: New York, Pennsylvania, New Jersey, Delaware, Maryland, West Virginia, Virginia, D.C., North Carolina, South Carolina, Louisianna, Tennessee, Mississippi, Alabama, Georgia, and Florida

West Coast:

The R.W. Walker Co., Inc.
Gordon Carnie
2716 Ocean Park Boulevard, Suite 1010,
Santa Monica, California 90405 (213) 450-9001

serving: Washington, Oregon, Idaho, Montana, Wyoming, Colorado, New Mexico, Arizona, Utah, Nevada, California, Alaska, and Hawaii (also British Columbia and Alberta, Canada)

Mid-West Territory:

Thomas Knorr & Associates
Thomas H. Knorr, Jr.
33 N. Michigan Avenue, Suite 403
Chicago, Illinois 60601 (312) 726-2633

serving: Ohio, Oklahoma, Arkansas, Texas, North Dakota, South Dakota, Nebraska, Kansas, Missouri, Indiana, Illinois, Iowa, Michigan, Wisconsin, and Minnesota.